AI Capstone Project 1 report

111550102 林鈺誠

1. Data link: https://github.com/vitolin0416/2025_AIC/tree/main/HW1
2. Research Question: The objective of this project is to **classify images of the five members of NJZ** using different machine learning methods such as HOG + SVM and CNN and others.

   The motivation of this project is that a lot of people cannot tell the difference between the memebers of k-pop girl group, while it is very easy for the fans to regonize them. For me, I was frustrated at the beginning, while I can tell the differences very easily right now. I think the main cause is that the makeups making each of them looks alike. While in different stages they come up with different makeups, making it harder to find the similarities in each members. This is an interesting topic, so I want to know that if the computers can "learn" to regonize them in a relatively small data set.
3. Dataset Documentation:

   Datatype: The main data type is image. To be more specific it is .jpg.

   Numbers: For each member, there are at least 180 photos. The total number of photos is 1125. (there are five members in NJZ!)

   Collection methods: The data is collected from internet, include fan made websites, social media, pinterest, and others. The data is downloaded by myself, I tried to use web scrawler but there are always some bugs, thus I quit.

   Data Preprocessing: Face Detection and Cropping using MTCNN. I use MTCNN (Multi-task Cascaded Convolutional Networks) to detect faces, once a face is detected, the script extract the photo with extra 20% padding to ensure that the cropped image contain some spaces around the face. This helps the regonition model to improve performance by reducing noises.
4. Methods Used:
   1. **HOG+SVM**: I use **HOG (Histogram of Oriented Gradients)**, a feature descriptor that counts occurrences of gradient orientation in localized portions of an image.180 photos per member is used. Given a overall 900 pictures data set.

      HOG parameters: Each image is resized to 64*64 pixel, and turned into

grayscale. And then using the hog function from the skimage.feature module with the following parameters:

- Orientations: 9 (The number of orientation bins for the gradient histograms)
- Pixels per cell: (8, 8) (The size of the cell over which gradients are computed)
- Cells per block: (2, 2) (The number of cells that make up a block, used for normalization)

SVM: And then I use Support Vector Machine, which is trained on the feature vectors from the HOG discription, to classify the images into one of the five members of NJZ.

Cross-Validation: The performance of the SVM classifier is assessed using **StratifiedKFold cross-validation**. This method splits the dataset into 5 parts, trains the model on 4 parts, and tests it on the remaining part.

Libraries Used:

OpenCV, skimage.feature, scikit-learn, Matplotlib

Results:

The data used is 180 randomly selected pictures for each member.

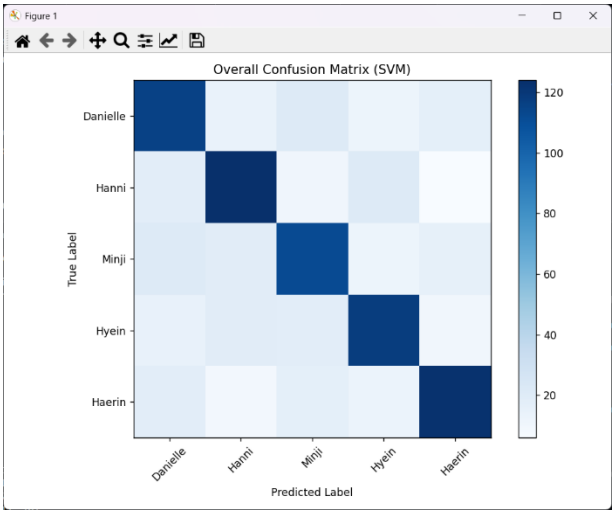
For 5 folds, the model use 36 pictures for testing each time.

Cross-validation scores: [0.68, 0.68, 0.64, 0.67, 0.61]

Mean CV accuracy: 0.658 ± 0.025

	Precision	Recall	F1-Score
Danielle	0.62	0.64	0.63
Hanni	0.67	0.69	0.68
Minji	0.63	0.62	0.62
Hyein	0.67	0.66	0.66
Haerin	0.72	0.68	0.70
accuracy			0.66
macro avg	0.66	0.66	0.66
weighted avg	0.66	0.66	0.66

Predicted label / True lable	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	116	14	21	12	17
Hanni	18	124	11	21	6
Minji	21	19	112	12	16
Hyein	15	19	18	118	10
Haerin	18	9	17	13	123



The result of HOG+SVM is quite impressive, even though its accuracy is less than 0.8. It still shows potential in a relatively small dataset.

2. **CNN:** Use the Convolutional Neural Network classifier is to classify the facial images of the five NJZ members. CNN can extract the features through convolution operations, without the feature extraction step.
   180 pictures for each member as dataset, given a total 900 pictures dataset.
   Data preprocessing: The images are resized to 128*128 pixels and normalized by scaling pixel values to the range [0, 1].
   CNN detail discription: Optimizer: Use Adam Optimizer to adapt the learning rate during training to improve convergence. Loss Function: categorical cross-entropy loss function is used.
   Cross-Validation: 5-fold Stratified Cross-Validation is used to evaluate the model's performance.
   Early Stopping: To prevent overfitting, stop the training when the validation accuracy does not improve for 5 consecutive epochs.
   Libraries Used:
   TensorFlow / Keras, OpenCV, NumPy, Matplotlib, Scikit-learn.

Results:

Cross-validation results:

Fold 1: 0.3722 Fold 2: 0.3333 Fold 3: 0.3944 Fold 4: 0.4500 Fold 5: 0.3444

Mean CV accuracy: 0.3789 ± 0.0415

Class	Precision	Recall	F1-Score
Danielle	0.28	0.11	0.16
Hanni	0.35	0.39	0.37
Minji	0.41	0.52	0.46
Hyein	0.36	0.52	0.43
Haerin	0.45	0.35	0.39
Accuracy			0.38
Macro avg	0.37	0.38	0.36
Weighted avg	0.37	0.38	0.36

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	20	37	53	53	17
Hanni	10	71	30	43	26
Minji	22	19	93	31	15
Hyein	13	34	20	94	19
Haerin	6	44	30	37	63

The performance of CNN is not good, since the accuracy is only 0.38, slightly better than 0.2 (pure guessing).

This might because of the differences between each member is not very large. Also the dataset is quite small, which is not very suitable for deap-learning models like CNN.

3. HOG+K-mean clustering: Use K-means clustering to group facial images and test if the algorithm can segregate different members based on the extracted features.

   Image preprocessing: Images are converted to grayscale and resized to 128x128 pixels to standardize the input.

   HOG: Same as the SVM part. The hog function from skimage.feature is

used to extract HOG features with the following parameters:

- Orientations: 9 (bins for gradient directions)
- Pixels per cell: (8, 8) (size of the cell for gradient calculation)
- Cells per block: (2, 2) (number of cells per block for normalization)

K-Means Clustering: K-Means attempts to partition the dataset into **5 clusters** (since there are 5 members) based on the feature similarities between images.

n_clusters: 5 (number of clusters, corresponding to the 5 members)

n_init: 10 (number of initializations of the centroid to avoid local minima)

Libraries Used:

OpenCV, NumPy, Matplotlib, scikit-learn, scipy, skimage.feature.

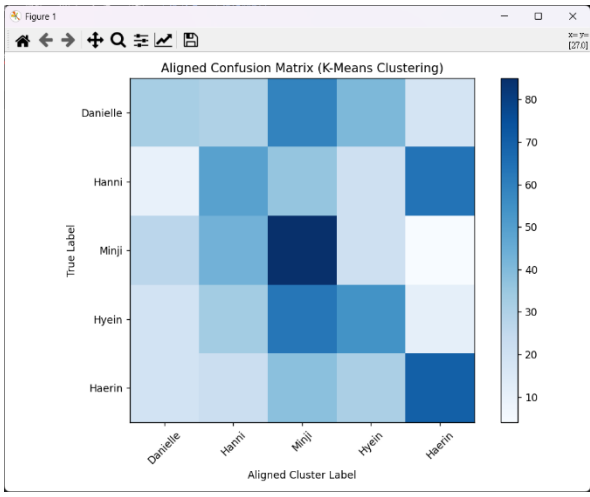
Result: I use ARI, NMI and Confusion Matrix with Hungarian algorithm to align the clusters with the true labels.

Adjusted Rand Index (ARI): 0.0484

Normalized Mutual Information (NMI): 0.0708

aligned confusion matrix:

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	32	30	59	41	18
Hanni	10	49	36	21	64
Minji	27	43	85	21	4
Hyein	19	33	63	54	11
Haerin	19	22	38	31	70



Clustering Accuracy (after alignment): 0.3222

The result of k-means clustering is not very good, either.

I think it is because of the HOG feature are alike for human faces, making it harder for the algorithm to distinguish and form different clusters.

5. Experiments:

1. Dataset quantaties:

   I perform HOG+SVM with the same parameters as above, with different dataset number (50, 100, 150, 180 per member)

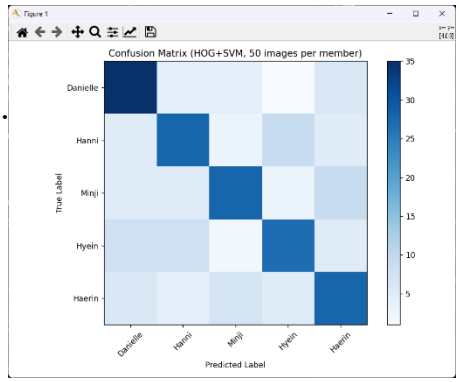
   For 50 images per member (total 250 pictures):

Cross-validation scores: [0.52, 0.68, 0.6, 0.58, 0.54]

Mean CV accuracy: 0.5840 ± 0.0557

Class	Precision	Recall	F1-Score
Danielle	0.64	0.56	0.60
Hanni	0.60	0.54	0.57
Minji	0.53	0.56	0.54
Hyein			0.58
Haerin	0.59	0.58	0.58
Accuracy	0.59	0.58	0.58
Macro avg	0.64	0.56	0.60
Weighted avg	0.60	0.54	0.57

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	35	4	4	1	6
Hanni	5	28	3	9	5
Minji	5	5	28	3	9
Hyein	8	8	2	27	5
Haerin	6	4	7	5	28



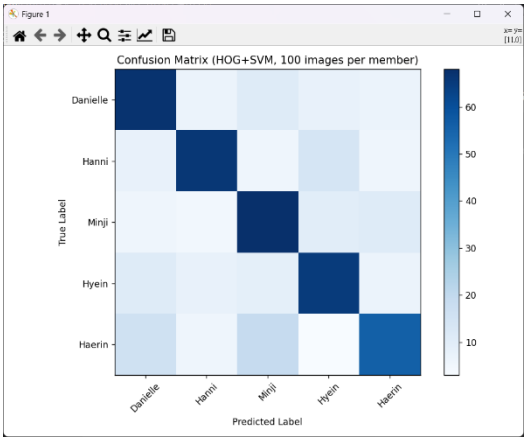
For 100 images per member (total 500 pictures):

Cross-validation scores: [0.61 0.61 0.62 0.68 0.7]

Mean CV accuracy: 0.6440 ± 0.0383

Class	Precision	Recall	F1-Score
Danielle	0.62	0.67	0.64
Hanni	0.72	0.66	0.69
Minji	0.60	0.68	0.64
Hyein	0.65	0.65	0.65
Haerin	0.64	0.56	0.60
Accuracy			**0.64**
Macro avg	0.65	0.64	0.64
Weighted avg	0.65	0.64	0.64

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	67	7	11	8	7
Hanni	8	66	6	14	6
Minji	6	5	68	10	11
Hyein	11	8	9	65	7
Haerin	16	6	19	3	56



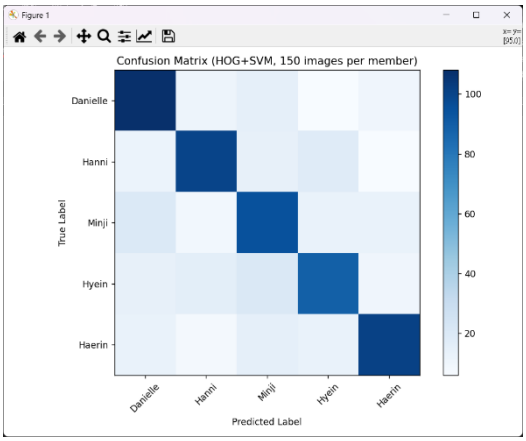
For 150 images per member (total 750 pictures):

Cross-validation scores: [0.6867, 0.6333, 0.6467, 0.66, 0.66]

Mean CV accuracy: 0.6573 ± 0.0177

Class	Precision	Recall	F1-Score
Danielle	0.65	0.72	0.68
Hanni	0.69	0.67	0.68
Minji	0.59	0.63	0.61
Hyein	0.64	0.59	0.62
Haerin	0.72	0.67	0.70
Accuracy			**0.66**
Macro avg	0.66	0.66	0.66
Weighted avg	0.66	0.66	0.66

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	108	11	15	6	10
Hanni	12	100	14	18	6
Minji	20	9	95	13	13
Hyein	14	16	21	89	10
Haerin	13	8	15	13	101



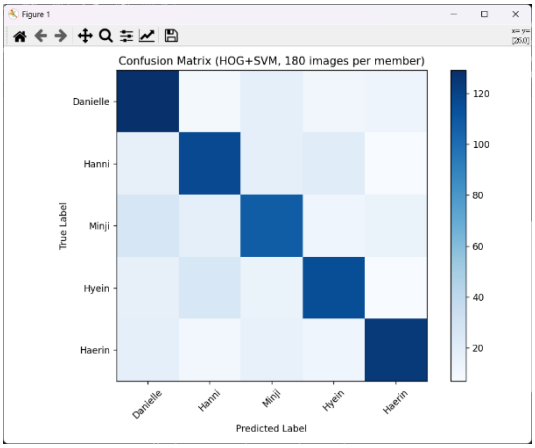
For 180 images per member (total 900 pictures):

Cross-validation scores: [0.6611, 0.6833, 0.6889, 0.6389, 0.6222]

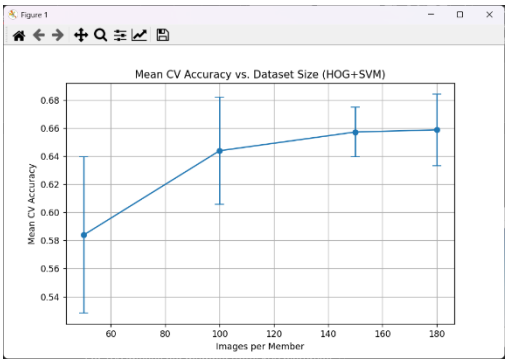
Mean CV accuracy: 0.6589 ± 0.0255

Class	Precision	Recall	F1-Score
Danielle	0.62	0.72	0.66
Hanni	0.65	0.65	0.65
Minji	0.62	0.60	0.61
Hyein	0.67	0.64	0.66
Haerin	0.75	0.69	0.72
Accuracy			**0.66**
Macro avg	0.66	0.66	0.66
Weighted avg	0.66	0.66	0.66

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	129	9	18	11	13
Hanni	17	117	18	21	7
Minji	27	18	108	12	15
Hyein	17	26	15	115	7
Haerin	18	10	16	12	124



As shown in this picture, the accuracy increases as the size of dataset increases. However, the magnitude of increase slows down when number of dataset changes from 150 to 180. I think that is because when the size become larger, you have to add more data than before (two times more, for instance) to see a significant change.



2. SMOTE oversampling:

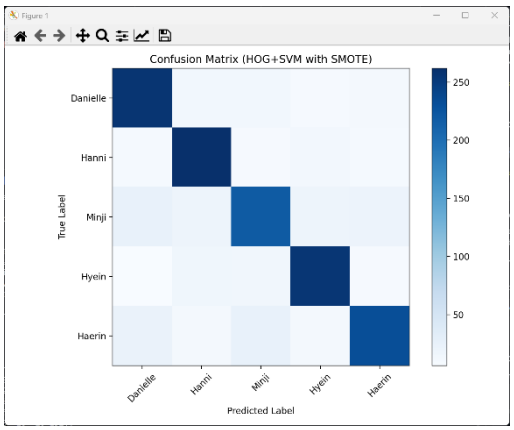
Since the dataset of mine is imbalanced (some member have photos more that 270). I use SMOTE to oversample every member's dataset to 300, resulting a total 1500 unit dataset.

Result: Cross-validation scores: [0.8233, 0.8167, 0.85, 0.83, 0.76]

Mean CV accuracy: 0.8160 ± 0.0301

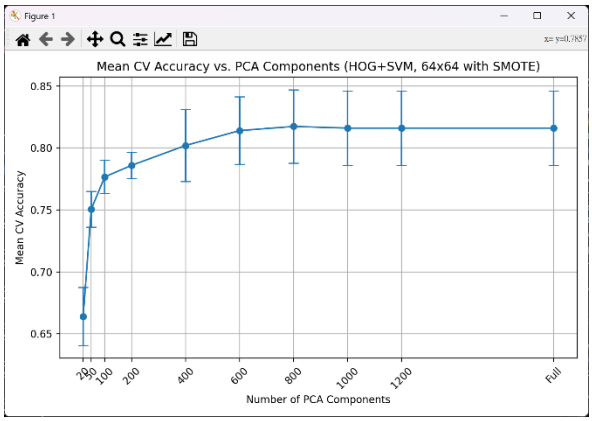
Class	Precision	Recall	F1-Score
Danielle	0.81	0.85	0.83
Hanni	0.82	0.87	0.85
Minji	0.78	0.73	0.75
Hyein	0.84	0.85	0.85
Haerin	0.83	0.77	0.80
Accuracy			**0.82**
Macro avg	0.82	0.82	0.82
Weighted avg	0.82	0.82	0.82

	Danielle	Hanni	Minji	Hyein	Haerin
Danielle	256	13	13	8	10
Hanni	9	262	8	12	9
Minji	24	18	219	18	21
Hyein	6	16	15	255	8
Haerin	22	10	26	10	232



The result is very good, compared to the original method (with 180 photos each member). This is expected because with oversampling we manage to use the whole data set and thus improve the accuracy.

3. PCA Components:



Use PCA to reduce the dimensions of features. Thus reduce the time needed for evaluation and remove some noises that might affect the performance.

I use SMOTE to oversampling to 300 data for each member, for a total of 1500 data points. Test differenr amount of PCA components: 20, 100, 200, 400, 600, 800, 1000, 1200, and full (1764)

The result shows that when the dimensions increases to a certain number, the accuracy remain approximately the same. This is useful when dataset is large since reduce dimensions can improve efficiency greatly.

6. Discussion:

1.Based on your experiments, are the results and observed behaviors what you expect?

Ans: Yes, the experiments results are similar from what I expected. However, I'm amazed by the power of SMOTE oversampling. At first I expected the accuracy will improve for like 0.05 at most. But it increased for 0.15, greater than 0.8. Which is way higher than I expected.

2.Discuss factors that affect the performance, including dataset characteristics.

Ans: The quantity, quality, diversity of dataset are very important. For this assignment, some photo's quality is relatively bad (with low resolution) or some photos are very alike (taked from the same activity/performance). Which I think also affect the performance greatly.

3.Describe experiments that you would do if there were more time available.

Ans: I would like to do experiments about different quality of pictures, for instance, limit and resize the images to 256*256 pixel, and see if the performance improve. However, this will increase the complexity of cauculation and increase time needed. I would also like to increase the number of dataset till the CNN method would work. But that would require a lot of data, which does not seems feasable.

4.Indicate what you have learned from the experiments as well as your remaining questions.

Ans: The experiment of PCA makes me realize that reduce the number of dimensions in a acceptable range would not reduce performance. In the future if I have to implement models with high-dimensional data, I would think of this and thus reduce the time needed for training.

I think that the remaining questions is that I still don't know how many photos need for CNN to perform normally. And also how many photos need for SVM to achieve a human like accuracy (for at least 0.95 would be nice).

Appendex:

The first one is the one I use to crop the photos (as preprocessing):

```
import os
import cv2
from mtcnn import MTCNN
import numpy as np
import shutil

# Root directories
input_root_dir = "Hyein_renamed_photo_dataset"   # Renamed dataset with ASCII names
output_root_dir = "NJZ_cropped_faces_dataset"   # Cropped faces
original_successful_dir = "NJZ_original_successful_dataset"   # Originals of successful crops

# List of members to process (modify this as needed)
members_to_process = ["Hyein"]   # Example: process Minji and Hanni only

# Supported image extensions
image_extensions = (".jpg", ".jpeg", ".png")

# Initialize MTCNN face detector
detector = MTCNN()

# Create output directories
if not os.path.exists(output_root_dir):
    os.makedirs(output_root_dir)
if not os.path.exists(original_successful_dir):
    os.makedirs(original_successful_dir)

# Function to generate a unique filename
def get_unique_filename(base_path, filename):
    base, ext = os.path.splitext(filename)   # Split into name and extension (e.g., "hanni_001", ".jpg")
    # Extract the original number (e.g., "001" from "hanni_001")
    prefix, num = base.rsplit("_", 1) if "_" in base and base.rsplit("_", 1)[-1].isdigit() else (base, "0")
    counter = 1
    new_filename = filename
    new_path = os.path.join(base_path, new_filename)

    while os.path.exists(new_path):
        new_filename = f"{prefix}_{counter}_{num}{ext}"   # e.g., "hanni_1_001.jpg"
        new_path = os.path.join(base_path, new_filename)
        counter += 1

    return new_filename

# Function to crop face from an image
def crop_face(image_path, output_path, original_output_path):
    # Read the image
    img = cv2.imread(image_path)
    if img is None:
        print(f"Failed to load {image_path}")
        return False

    # Convert to RGB (MTCNN expects RGB)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Detect faces
    faces = detector.detect_faces(img_rgb)

    if len(faces) == 0:
        print(f"No face detected in {image_path}")
        return False

    # Use the first detected face (assume one face per image)
    face = faces[0]
    x, y, w, h = face['box']

    # Add padding around the face (20% of the bounding box size)
    padding = int(max(w, h) * 0.2)
```

```
    x = max(0, x - padding)
    y = max(0, y - padding)
    w = min(img.shape[1] - x, w + 2 * padding)
    h = min(img.shape[0] - y, h + 2 * padding)

    # Crop the face
    cropped = img[y:y+h, x:x+w]

    # Save the cropped image with unique name
    unique_output_filename = get_unique_filename(os.path.dirname(output_path), os.path.basename(output_path))
    unique_output_path = os.path.join(os.path.dirname(output_path), unique_output_filename)
    cv2.imwrite(unique_output_path, cropped)

    # Copy the original image with unique name
    unique_original_filename = get_unique_filename(os.path.dirname(original_output_path),
os.path.basename(original_output_path))
    unique_original_output_path = os.path.join(os.path.dirname(original_output_path), unique_original_filename)
    shutil.copy2(image_path, unique_original_output_path)

    if unique_output_filename != os.path.basename(output_path):
        print(f"   Renamed {os.path.basename(output_path)} to {unique_output_filename} in cropped folder")
    if unique_original_filename != os.path.basename(original_output_path):
        print(f"   Renamed {os.path.basename(original_output_path)} to {unique_original_filename} in original folder")

    return True

# Process specified members
total_successful_crops = 0
total_failed_crops = 0

for member_to_process in members_to_process:
    member_input_path = os.path.join(input_root_dir, member_to_process)
    member_output_path = os.path.join(output_root_dir, member_to_process)
    original_member_output_path = os.path.join(original_successful_dir, member_to_process)

    if os.path.isdir(member_input_path):
        # Create output folders
        if not os.path.exists(member_output_path):
            os.makedirs(member_output_path)
        if not os.path.exists(original_member_output_path):
            os.makedirs(original_member_output_path)

        print(f"Processing {member_to_process}...")
        total_images = 0
        successful_crops = 0
        failed_crops = 0

        # Loop through images
        for filename in os.listdir(member_input_path):
            if filename.lower().endswith(image_extensions):
                total_images += 1
                input_path = os.path.join(member_input_path, filename)
                output_path = os.path.join(member_output_path, filename)
                original_output_path = os.path.join(original_member_output_path, filename)

                # Crop and save the face
                if crop_face(input_path, output_path, original_output_path):
                    successful_crops += 1
                else:
                    failed_crops += 1

        # Update global counts
        total_successful_crops += successful_crops
        total_failed_crops += failed_crops

        # Print results for this member
        print(f"   {member_to_process}: {successful_crops}/{total_images} images cropped successfully")
        print(f"   Successful crops: {successful_crops}")
        print(f"   Failed crops: {failed_crops}\n")
    else:
        print(f"Folder {member_to_process} not found in {input_root_dir}")
```

```
# Print overall results
print(f"Overall results:")
print(f"   Total successful crops: {total_successful_crops}")
print(f"   Total failed crops: {total_failed_crops}")
print("Face cropping completed!")
```

The second one is the code for HOG+SVM

```
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predict
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import os
from glob import glob
import random
import shutil

# Set random seed for reproducibility
random_seed = 722
random.seed(random_seed)
np.random.seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]

# Store images and labels
images = []
labels = []
file_paths = []

# Load images for each member
for idx, member in enumerate(members):
    member_path = os.path.join(dataset_path, member)
    image_paths = glob(os.path.join(member_path, "*.jpg"))

    # Randomly select 180 images per member
    if len(image_paths) > 180:
        image_paths = random.sample(image_paths, 180)

    for img_path in image_paths:
        # Read image
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)   # Convert to grayscale
        img = cv2.resize(img, (64, 64))   # Resize to consistent dimensions

        # Append image, label, and file path
        images.append(img)
        labels.append(idx)
        file_paths.append(img_path)

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
file_paths = np.array(file_paths)

# Extract HOG features
print("Extracting HOG features...")
hog_features = []
for image in images:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2), visualize=False)
    hog_features.append(features)
```

```
hog_features = np.array(hog_features)

# Perform cross-validation
print("Performing 5-fold cross-validation...")
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
svm = SVC(kernel='linear', probability=True, random_state=random_seed)

# Cross-validation scores (accuracy)
cv_scores = cross_val_score(svm, hog_features, labels, cv=cv, scoring='accuracy')
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

# After computing cv_scores
from sklearn.model_selection import cross_val_predict

# Get predictions across all folds
y_pred_all = cross_val_predict(svm, hog_features, labels, cv=cv)

# Overall classification report
print("\nOverall Classification Report:")
print(classification_report(labels, y_pred_all, target_names=members))

# Overall confusion matrix
overall_cm = confusion_matrix(labels, y_pred_all)
print("\nOverall Confusion Matrix:")
print(overall_cm)

# Visualize
plt.figure(figsize=(8, 6))
plt.imshow(overall_cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Overall Confusion Matrix (SVM)")
plt.colorbar()
tick_marks = np.arange(len(members))
plt.xticks(tick_marks, members, rotation=45)
plt.yticks(tick_marks, members)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```

the third one is for CNN:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
import random
import shutil
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping

# Set random seed for reproducibility
random_seed = 42
random.seed(random_seed)
np.random.seed(random_seed)
tf.random.set_seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]
```

```
# Store images and labels
images = []
labels = []
file_paths = []

# Load images for each member
for idx, member in enumerate(members):
    member_path = os.path.join(dataset_path, member)
    image_paths = glob(os.path.join(member_path, "*.jpg"))

    # Randomly select 180 images per member
    if len(image_paths) > 180:
        image_paths = random.sample(image_paths, 180)

    for img_path in image_paths:
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (128, 128))
        images.append(img)
        labels.append(idx)
        file_paths.append(img_path)

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
file_paths = np.array(file_paths)

# Normalize pixel values to 0-1
images = images / 255.0

# Function to create CNN model
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(128, 128, 3)),
        Conv2D(32, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Dropout(0.25),
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(5, activation='softmax')
    ])

    model.compile(
        optimizer=Adam(learning_rate=0.0001),
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# Perform cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
cv_scores = []
all_y_true = []
all_y_pred = []

print("Performing 5-fold cross-validation...")
for fold, (train_idx, val_idx) in enumerate(cv.split(images, labels)):
    print(f"\nFold {fold+1}/5")

    # Split data
    X_train_fold, X_val_fold = images[train_idx], images[val_idx]
    y_train_fold, y_val_fold = labels[train_idx], labels[val_idx]
```

```
    # Convert to one-hot encoding
    y_train_fold_onehot = to_categorical(y_train_fold, num_classes=5)
    y_val_fold_onehot = to_categorical(y_val_fold, num_classes=5)

    # Create data augmentation
    datagen = ImageDataGenerator(
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )

    # Create and train model
    model = create_model()

    # Early stopping
    early_stopping = EarlyStopping(
        monitor='val_accuracy',
        patience=5,
        restore_best_weights=True
    )

    # Train model
    history = model.fit(
        datagen.flow(X_train_fold, y_train_fold_onehot, batch_size=32),
        epochs=15,
        validation_data=(X_val_fold, y_val_fold_onehot),
        callbacks=[early_stopping],
        verbose=1
    )

    # Evaluate model
    val_loss, val_acc = model.evaluate(X_val_fold, y_val_fold_onehot, verbose=0)
    cv_scores.append(val_acc)
    print(f"Validation accuracy: {val_acc:.4f}")

    # Get predictions for this fold
    y_pred_proba = model.predict(X_val_fold)
    y_pred = np.argmax(y_pred_proba, axis=1)   # Convert probabilities to class labels
    y_true = y_val_fold   # True labels (not one-hot encoded)

    # Store true and predicted labels for overall metrics
    all_y_true.extend(y_true)
    all_y_pred.extend(y_pred)

    # Per-fold classification report
    print(f"\nClassification Report for Fold {fold+1}:")
    print(classification_report(y_true, y_pred, target_names=members))

    # Per-fold confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    print(f"Confusion Matrix for Fold {fold+1}:\n{cm}")

# Print cross-validation results
print("\nCross-validation results:")
for i, score in enumerate(cv_scores):
    print(f"Fold {i+1}: {score:.4f}")
print(f"Mean CV accuracy: {np.mean(cv_scores):.4f} ± {np.std(cv_scores):.4f}")

# Overall classification report across all folds
print("\nOverall Classification Report:")
print(classification_report(all_y_true, all_y_pred, target_names=members))

# Overall confusion matrix across all folds
overall_cm = confusion_matrix(all_y_true, all_y_pred)
print("\nOverall Confusion Matrix:")
print(overall_cm)
```

```
# Optional: Visualize the overall confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(overall_cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Overall Confusion Matrix")
plt.colorbar()
tick_marks = np.arange(len(members))
plt.xticks(tick_marks, members, rotation=45)
plt.yticks(tick_marks, members)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```

the fourth one is for K-means:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from glob import glob
import random
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score, confusion_matrix
from scipy.optimize import linear_sum_assignment
from skimage.feature import hog

# Set random seed for reproducibility
random_seed = 722
random.seed(random_seed)
np.random.seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]

# Store images and labels
images = []
labels = []
file_paths = []

# Load images for each member
for idx, member in enumerate(members):
    member_path = os.path.join(dataset_path, member)
    image_paths = glob(os.path.join(member_path, "*.jpg"))

    # Randomly select 180 images per member
    if len(image_paths) > 180:
        image_paths = random.sample(image_paths, 180)

    for img_path in image_paths:
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (128, 128))
        images.append(img)
        labels.append(idx)
        file_paths.append(img_path)

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
file_paths = np.array(file_paths)

# Extract HOG features
print("Extracting HOG features...")
hog_features = []
for image in images:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2), visualize=False)
    hog_features.append(features)
```

```
hog_features = np.array(hog_features)

# Perform K-Means clustering
print("Performing K-Means clustering...")
kmeans = KMeans(n_clusters=5, random_state=random_seed, n_init=10)
cluster_labels = kmeans.fit_predict(hog_features)

# Evaluation metrics
ari = adjusted_rand_score(labels, cluster_labels)
nmi = normalized_mutual_info_score(labels, cluster_labels)

print(f"\nEvaluation Metrics:")
print(f"Adjusted Rand Index (ARI): {ari:.4f}")
print(f"Normalized Mutual Information (NMI): {nmi:.4f}")

# Compute initial confusion matrix (before alignment)
cm_initial = confusion_matrix(labels, cluster_labels)
print("\nInitial Confusion Matrix (True Labels vs. Cluster Labels, Unaligned):")
print(cm_initial)

# Align cluster labels with true labels using the Hungarian algorithm
def align_labels(true_labels, cluster_labels):
    cm = confusion_matrix(true_labels, cluster_labels)
    row_ind, col_ind = linear_sum_assignment(-cm)   # Maximize diagonal by minimizing -cm
    label_mapping = {old_label: new_label for old_label, new_label in zip(col_ind, row_ind)}
    aligned_labels = np.array([label_mapping[label] for label in cluster_labels])
    return aligned_labels

# Get aligned cluster labels
aligned_cluster_labels = align_labels(labels, cluster_labels)

# Compute aligned confusion matrix
cm_aligned = confusion_matrix(labels, aligned_cluster_labels)
print("\nAligned Confusion Matrix (True Labels vs. Aligned Cluster Labels):")
print(cm_aligned)

# Compute clustering accuracy based on aligned labels
clustering_accuracy = np.sum(np.diag(cm_aligned)) / np.sum(cm_aligned)
print(f"\nClustering Accuracy (after alignment): {clustering_accuracy:.4f}")

# Visualize the aligned confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm_aligned, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Aligned Confusion Matrix (K-Means Clustering)")
plt.colorbar()
tick_marks = np.arange(len(members))
plt.xticks(tick_marks, members, rotation=45)
plt.yticks(tick_marks, members)
plt.ylabel('True Label')
plt.xlabel('Aligned Cluster Label')
plt.tight_layout()
plt.show()

# Analyze cluster composition
print("\nCluster Composition (Number of samples per cluster):")
unique, counts = np.unique(cluster_labels, return_counts=True)
for cluster, count in zip(unique, counts):
    print(f"Cluster {cluster}: {count} samples")
```

the fifth one is for experiment one, dataset difference:

```
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predict
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import os
from glob import glob
import random
```

```
# Set random seed for reproducibility
random_seed = 722
random.seed(random_seed)
np.random.seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]

# Dataset sizes to experiment with (images per member)
dataset_sizes = [50, 100, 150, 180]

# Function to load data with a specified size per member
def load_data(size_per_member):
    images = []
    labels = []
    file_paths = []

    for idx, member in enumerate(members):
        member_path = os.path.join(dataset_path, member)
        image_paths = glob(os.path.join(member_path, "*.jpg"))

        # Randomly select the specified number of images per member
        if len(image_paths) > size_per_member:
            image_paths = random.sample(image_paths, size_per_member)

        for img_path in image_paths:
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (64,64))
            images.append(img)
            labels.append(idx)
            file_paths.append(img_path)

    return np.array(images), np.array(labels), np.array(file_paths)

# Function to extract HOG features
def extract_hog_features(images):
    print("Extracting HOG features...")
    hog_features = []
    for image in images:
        features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                       cells_per_block=(2, 2), visualize=False)
        hog_features.append(features)
    return np.array(hog_features)

# Store results
results = {}

# Perform experiment for each dataset size
for size in dataset_sizes:
    print(f"\n=== Experiment with {size} images per member (Total: {size * len(members)} images) ===")

    # Load data for this size
    images, labels, file_paths = load_data(size)

    # Extract HOG features
    hog_features = extract_hog_features(images)

    # Perform cross-validation
    print("Performing 5-fold cross-validation...")
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
    svm = SVC(kernel='linear', probability=True, random_state=random_seed)

    # Get cross-validation accuracy scores
    cv_scores = cross_val_score(svm, hog_features, labels, cv=cv, scoring='accuracy')
    print(f"Cross-validation scores: {cv_scores}")
    print(f"Mean CV accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

    # Get predictions for detailed metrics
```

```
    y_pred = cross_val_predict(svm, hog_features, labels, cv=cv)

    # Compute and print classification report
    print("\nClassification Report:")
    report = classification_report(labels, y_pred, target_names=members, output_dict=True)
    print(classification_report(labels, y_pred, target_names=members))

    # Compute confusion matrix
    cm = confusion_matrix(labels, y_pred)
    print("\nConfusion Matrix:")
    print(cm)

    # Store results
    results[size] = {
        'accuracy': cv_scores.mean(),
        'std': cv_scores.std(),
        'precision': [report[member]['precision'] for member in members],
        'recall': [report[member]['recall'] for member in members],
        'f1': [report[member]['f1-score'] for member in members],
        'confusion_matrix': cm
    }

    # Visualize confusion matrix
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f"Confusion Matrix (HOG+SVM, {size} images per member)")
    plt.colorbar()
    tick_marks = np.arange(len(members))
    plt.xticks(tick_marks, members, rotation=45)
    plt.yticks(tick_marks, members)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.tight_layout()
    plt.show()

# Summarize results
print("\n=== Summary of Results ===")
for size in dataset_sizes:
    print(f"\nDataset Size: {size} images per member (Total: {size * len(members)})")
    print(f"Mean CV Accuracy: {results[size]['accuracy']:.4f} ± {results[size]['std']:.4f}")
    print("Per-class Precision:", [f"{x:.4f}" for x in results[size]['precision']])
    print("Per-class Recall:", [f"{x:.4f}" for x in results[size]['recall']])
    print("Per-class F1-score:", [f"{x:.4f}" for x in results[size]['f1']])

# Plot accuracy vs. dataset size
plt.figure(figsize=(8, 5))
accuracies = [results[size]['accuracy'] for size in dataset_sizes]
stds = [results[size]['std'] for size in dataset_sizes]
plt.errorbar(dataset_sizes, accuracies, yerr=stds, fmt='-o', capsize=5)
plt.title("Mean CV Accuracy vs. Dataset Size (HOG+SVM)")
plt.xlabel("Images per Member")
plt.ylabel("Mean CV Accuracy")
plt.grid(True)
plt.show()
```

the sixth one is for experiment two, SMOTE oversampling:

```
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predict
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import os
from glob import glob
import random

# Set random seed for reproducibility
random_seed = 722
```

```
random.seed(random_seed)
np.random.seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]

# Target size for oversampling (based on largest class, e.g., Minji/Haerin)
target_size_per_member = 300

# Store images and labels
images = []
labels = []
file_paths = []

# Load all available images for each member
for idx, member in enumerate(members):
    member_path = os.path.join(dataset_path, member)
    image_paths = glob(os.path.join(member_path, "*.jpg"))

    # No cap; use all available images
    print(f"Loaded {len(image_paths)} images for {member}")

    for img_path in image_paths:
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (64,64))
        images.append(img)
        labels.append(idx)
        file_paths.append(img_path)

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
file_paths = np.array(file_paths)

# Extract HOG features
print("\nExtracting HOG features...")
hog_features = []
for image in images:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2), visualize=False)
    hog_features.append(features)

hog_features = np.array(hog_features)

# Apply SMOTE to balance the dataset
print(f"\nApplying SMOTE to oversample to {target_size_per_member} images per member...")
smote = SMOTE(sampling_strategy={i: target_size_per_member for i in range(len(members))},
              random_state=random_seed)
hog_features_smote, labels_smote = smote.fit_resample(hog_features, labels)

# Check new dataset size
print(f"New dataset size after SMOTE: {len(labels_smote)} samples")
unique, counts = np.unique(labels_smote, return_counts=True)
for member_idx, count in zip(unique, counts):
    print(f"{members[member_idx]}: {count} samples")

# Perform cross-validation on SMOTE-balanced data
print("\nPerforming 5-fold cross-validation on SMOTE-balanced data...")
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
svm = SVC(kernel='linear', probability=True, random_state=random_seed)

# Get cross-validation accuracy scores
cv_scores = cross_val_score(svm, hog_features_smote, labels_smote, cv=cv, scoring='accuracy')
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

# Get predictions for detailed metrics
y_pred = cross_val_predict(svm, hog_features_smote, labels_smote, cv=cv)
```

```
# Compute and print classification report
print("\nClassification Report:")
report = classification_report(labels_smote, y_pred, target_names=members, output_dict=True)
print(classification_report(labels_smote, y_pred, target_names=members))

# Compute confusion matrix
cm = confusion_matrix(labels_smote, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Visualize confusion matrix
plt.figure(figsize=(8, 6))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title("Confusion Matrix (HOG+SVM with SMOTE)")
plt.colorbar()
tick_marks = np.arange(len(members))
plt.xticks(tick_marks, members, rotation=45)
plt.yticks(tick_marks, members)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()

# Summarize per-class metrics
print("\nPer-class Metrics Summary:")
print("Precision:", [f"{report[member]['precision']:.4f}" for member in members])
print("Recall:", [f"{report[member]['recall']:.4f}" for member in members])
print("F1-score:", [f"{report[member]['f1-score']:.4f}" for member in members])
```

the seventh one is for experiment three, pca:

```
import cv2
import numpy as np
from skimage.feature import hog
from sklearn.model_selection import StratifiedKFold, cross_val_score, cross_val_predict
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import os
from glob import glob
import random

# Set random seed for reproducibility
random_seed = 722
random.seed(random_seed)
np.random.seed(random_seed)

# Path to your dataset
dataset_path = "NJZ_cropped_faces_dataset"

# Define the members
members = ["Danielle", "Hanni", "Minji", "Hyein", "Haerin"]

# PCA components to test
# pca_components = [50, 100, 200, 400, 800, 1600, 3200, None]   # Expanded list
pca_components = [20, 50, 100, 200, 400, 600, 800, 1000, 1200, None]
# Target size for oversampling (based on largest class, e.g., Minji/Haerin)
target_size_per_member = 300

# Store images and labels
images = []
labels = []
file_paths = []

# Load all available images for each member
for idx, member in enumerate(members):
    member_path = os.path.join(dataset_path, member)
    image_paths = glob(os.path.join(member_path, "*.jpg"))
```

```
    # Use all available images (no cap)
    print(f"Loaded {len(image_paths)} images for {member}")

    for img_path in image_paths:
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (64, 64))
        images.append(img)
        labels.append(idx)
        file_paths.append(img_path)

# Convert to numpy arrays
images = np.array(images)
labels = np.array(labels)
file_paths = np.array(file_paths)

# Extract HOG features
print("\nExtracting HOG features...")
hog_features = []
for image in images:
    features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2), visualize=False)
    hog_features.append(features)

hog_features = np.array(hog_features)
print(f"Original HOG feature dimensionality: {hog_features.shape[1]}")

# Apply SMOTE to balance the dataset
print(f"\nApplying SMOTE to oversample to {target_size_per_member} images per member...")
smote = SMOTE(sampling_strategy={i: target_size_per_member for i in range(len(members))},
              random_state=random_seed)
hog_features_smote, labels_smote = smote.fit_resample(hog_features, labels)

# Check new dataset size
print(f"New dataset size after SMOTE: {len(labels_smote)} samples")
unique, counts = np.unique(labels_smote, return_counts=True)
for member_idx, count in zip(unique, counts):
    print(f"{members[member_idx]}: {count} samples")

# Store results
results = {}

# Perform experiment for each PCA setting
for n_components in pca_components:
    if n_components is None:
        print(f"\n=== Experiment with full HOG features (no PCA, {hog_features_smote.shape[1]} components) ===")
        features = hog_features_smote
    else:
        print(f"\n=== Experiment with PCA ({n_components} components) ===")
        pca = PCA(n_components=n_components, random_state=random_seed)
        features = pca.fit_transform(hog_features_smote)
        print(f"Explained variance ratio: {np.sum(pca.explained_variance_ratio_):.4f}")

    # Perform cross-validation
    print("Performing 5-fold cross-validation...")
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_seed)
    svm = SVC(kernel='linear', probability=True, random_state=random_seed)

    # Get cross-validation accuracy scores
    cv_scores = cross_val_score(svm, features, labels_smote, cv=cv, scoring='accuracy')
    print(f"Cross-validation scores: {cv_scores}")
    print(f"Mean CV accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

    # Get predictions for detailed metrics (optional, but not plotted)
    y_pred = cross_val_predict(svm, features, labels_smote, cv=cv)

    # Compute and print classification report (optional, for reference)
    print("\nClassification Report:")
    print(classification_report(labels_smote, y_pred, target_names=members))

    # Store results
```

```
    results[n_components if n_components is not None else 'full'] = {
        'accuracy': cv_scores.mean(),
        'std': cv_scores.std()
    }

# Summarize results
print("\n=== Summary of Results ===")
for n_components in results:
    print(f"\nPCA Components: {n_components}")
    print(f"Mean CV Accuracy: {results[n_components]['accuracy']:.4f} ± {results[n_components]['std']:.4f}")

# Plot accuracy vs. PCA components
plt.figure(figsize=(8, 5))
components_list = [n if n is not None else hog_features_smote.shape[1] for n in pca_components]
accuracies = [results[n if n is not None else 'full']['accuracy'] for n in pca_components]
stds = [results[n if n is not None else 'full']['std'] for n in pca_components]
plt.errorbar(components_list, accuracies, yerr=stds, fmt='-o', capsize=5)
plt.title("Mean CV Accuracy vs. PCA Components (HOG+SVM, 64x64 with SMOTE)")
plt.xlabel("Number of PCA Components")
plt.ylabel("Mean CV Accuracy")
plt.grid(True)
plt.xticks(components_list, [str(n) if n is not None else 'Full' for n in pca_components], rotation=45)
plt.tight_layout()
plt.show()
```

=== Thanks TAs! ===

Minji, Hanni, Danielle, Haerin, Hyein