For the problem, the first step was to be able to extract the middle frames of every training video, and then ideally use the same process to extract the middle frames of every test video. This can be done so by looping through each respective directories' videos and saving the middle frames in a new directory.

```
for video in os.listdir(directory):
    # only process video files
    if not video.lower().endswith((".mp4", ".avi", ".mov", ".mkv")):
        print(f"Skipping non-video file: {video}")
        continue

    video_path = os.path.join(directory, video)
    print(f"Processing training video: {video_path}")

    frameExtractor(video_path, frames_dir, i)

    frame_path = os.path.join(frames_dir, f"{i+1:05d}.png")
    img = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
```

After saving each frame, the code will use the provided HandShapeFeatureExtractor class to extract the features of the gesture in the frame. Then it will save the vectors it returns to a features dictionary that will be used later for cosine similarity.

```
for video in os.listdir(directory):
    # only process video files
    if not video.lower().endswith((".mp4", ".avi", ".mov", ".mkv")):
        print(f"Skipping non-video file: {video}")
        continue

    video_path = os.path.join(directory, video)
    print(f"Processing training video: {video_path}")

    frameExtractor(video_path, frames_dir, i)

    frame_path = os.path.join(frames_dir, f"{i+1:05d}.png")
    img = cv2.imread(frame_path, cv2.IMREAD_GRAYSCALE)
```

Lastly, when comparing the two sets of frames, it will need to properly generate the output label after comparison. It will map the result to a gesture based on when it created our results dictionaries. It will also compare the average scores to try and better our accuracy.

```
def compare_and_save_results(train_features, test_features, output_csv="Results.csv"):
    # Mapping gestures to numeric labels
    gesture_to_label = {
        "0": 0, "1": 1, "2": 2, "3": 3, "4": 4,
        "5": 5, "6": 6, "7": 7, "8": 8, "9": 9,
        "Decrease Fan Speed": 10,
        "FanOff": 11,
        "FanOn": 12,
        "Increase Fan Speed": 13,
        "LightOff": 14,
        "LightOn": 15,
        "SetThermo": 16
    }

    output_labels = []

    for test_name, test_vec in test_features.items():

        class_scores = {}
        best_match, best_score = None, -1
        #print(f"\n=== Comparing {test_name} ===")
        for train_name, train_vec in train_features.items():
            gesture_name = get_gesture_name(train_name)
            score = cosine_similarity(test_vec, train_vec)
            # #print(f"  vs {train_name}: {score:.4f}")
            if gesture_name not in class_scores:
                class_scores[gesture_name] = []
            class_scores[gesture_name].append(score)

        avg_scores = {g: np.mean(scores) for g, scores in class_scores.items()}

        best_gesture = max(avg_scores, key=avg_scores.get)
        label = gesture_to_label.get(best_gesture, -1)  # -1 if something unexpected
        output_labels.append(label)

    # Save single-column CSV
    with open(output_csv, "w", newline="") as f:
        writer = csv.writer(f)
        for label in output_labels:
            writer.writerow([label])

    print(f"Results saved to {output_csv}")
```

The two vectors will be normalized before they are compared to try and help mitigate inaccuracies.

```
def cosine_similarity(vect1, vect2):
    vect1 = vect1.flatten()
    vect2 = vect2.flatten()
    vect1 = vect1 / (np.linalg.norm(vect1) + 1e-10)
    vect2 = vect2 / (np.linalg.norm(vect2) + 1e-10)
    return np.dot(vect1, vect2)
```

The following method is used to also help map the proper names based upon the training videos file name.

```
def get_gesture_name(filename):
    name = os.path.splitext(os.path.basename(filename))[0]  # remove extension
    if name.startswith("H-"):
        name = name[2:]  # remove "H-"

    # Map exact names to formatted labels
    mapping = {
        "DecreaseFanSpeed": "Decrease Fan Speed",
        "FanOff": "FanOff",
        "FanOn": "FanOn",
        "IncreaseFanSpeed": "Increase Fan Speed",
        "LightOff": "LightOff",
        "LightOn": "LightOn",
        "SetThermo": "SetThermo"
    }

    if name in mapping:
        return mapping[name]

    # Otherwise it's a digit 0-9
    return name
```

For the solution, the training directory will be the “traindata” directory, and test directory will be the “test” directory. Afterwards, process_all_training_videos and process_all_testing_videos will be called to obtain the middle frames of each respective directories’ videos before performing feature extraction on them. Lastly, compare_and_save_results will compare the test videos to the training videos to write into the “Results.csv” their output label.