ĐẠI HỌC ĐÀ NẴNG

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN TỬ VIỄN THÔNG

Trí Tuệ Nhân Tạo

ĐỀ TÀI: HỆ THỐNG AI TỰ ĐỘNG PHÂN LOẠI LÁ CÂY

SVTH : TRẦN THANH KHOA

THÁI ĐỨC TOÀN

MAI ĐỨC KHIÊM

NHÓM: 11

ĐÀ NẴNG , 2024

PHỤ LỤC

BẢNG ĐÁNH GIÁ MỨC ĐỘ ĐÓNG GÓP CÔNG VIỆC

Sinh viên	Mai Đức Khiêm	Trần Thanh Khoa	Thái Đức Toàn
% đóng góp	40 (KNN và ANN với HOG)	40 (KNN và ANN với Hu’s Moments)	20 (Phần 1)

NỘI DUNG CÔNG VIỆC:

Phần 1: Giới thiệu đề tài

Phần 2: Phương pháp thực hiện

Quá trình thực hiện bao gồm các bước chính:

- Bước 1: Chuẩn bị dữ liệu lá cây từ 4 đến 5 loài, tương tự như bài tập lớn trước.
- Bước 2: Trích xuất đặc trưng Hu’s moment và HOG từ ảnh lá cây, lưu trữ dưới dạng bảng Excel hoặc CSV để phục vụ cho việc phân tích và huấn luyện mô hình.
- Bước 3: Sử dụng thuật toán KNN để phân loại lá cây, thay đổi giá trị K và đánh giá hiệu quả hệ thống thông qua 5-fold cross-validation.
- Bước 4: Áp dụng phương pháp ANN với một lớp ẩn, hàm sigmoid và tốc độ học 0.05. Thay đổi số lượng nơ-ron trong lớp ẩn để khảo sát ảnh hưởng đến hiệu quả hệ thống, đồng thời sử dụng 5-fold cross-validation để đánh giá.
- Bước 5: So sánh hiệu quả giữa các đặc trưng (Hu’s moment và HOG) và giữa hai thuật toán KNN, ANN, từ đó đưa ra nhận xét.

Phần 3: Kết luận và hướng phát triển

Cuối cùng, hệ thống phân loại lá cây được đánh giá dựa trên kết quả thử nghiệm với cả hai phương pháp KNN và ANN. Hướng phát triển bao gồm việc tối ưu hoá mô hình, mở rộng bộ dữ liệu và ứng dụng các phương pháp học sâu để nâng cao độ chính xác trong phân loại.

CHƯƠNG 1: GIỚI THIỆU CHUNG

1. Tính cấp thiết

Trong bối cảnh hiện nay, việc ứng dụng công nghệ AI trong nông nghiệp đang ngày càng trở nên phổ biến và cần thiết. Đặc biệt, phân loại lá cây là một vấn đề quan trọng nhằm hỗ trợ việc nhận diện các loài cây trồng, phát hiện sâu bệnh, và quản lý cây xanh. Tuy nhiên, quá trình phân loại này thường đòi hỏi sự tham gia của các chuyên gia, mất nhiều thời gian và chi phí. Do đó, việc xây dựng hệ thống AI tự động phân loại lá cây không chỉ giúp giảm thiểu thời gian và công sức của con người mà còn tăng tính chính xác và hiệu quả trong việc nhận diện và quản lý các loài thực vật.

2. Giới thiệu phương pháp sử dụng

Hệ thống phân loại lá cây này sử dụng các đặc trưng hình ảnh tiên tiến và thuật toán học máy (ML) nhằm nhận diện và phân loại các loài lá khác nhau dựa trên hình dạng, kết cấu và màu sắc của chúng. Một trong những phương pháp phổ biến để trích xuất đặc trưng từ hình ảnh lá cây là sử dụng Moment Invariants (HU's Moments). Các đặc trưng HU's Moments, được giới thiệu vào năm 1962 là một bộ các đặc trưng hình học bất biến với các phép biến đổi hình học như quay, dịch chuyển và tỷ lệ. Một phương pháp khác để trích xuất đặc trưng là Histogram of Oriented Gradients (HOG), được phát triển và công bố trong tài liệu. Đặc trưng HOG tập trung vào việc phân tích sự phân bố của các gradient hướng trong hình ảnh, giúp nắm bắt các chi tiết hình dạng và cấu trúc phức tạp.

Đặc trưng hình ảnh:

- Moment Hu: Bộ 7 đặc trưng hình học không đổi theo phép quay và tỉ lệ, giúp mô tả hình dạng của lá cây.
- Histogram of Oriented Gradients (HOG): Đặc trưng dựa trên hướng gradient giúp mô tả kết cấu và biên dạng của lá.

Thuật toán học máy: Hệ thống sử dụng mạng nơ-ron nhân tạo (Artificial Neural Network - ANN) mô phỏng cách hoạt động của não bộ con người để xử lý và học từ dữ liệu. Đây là một mô hình phổ biến trong học sâu (Deep Learning) để nhận dạng các mẫu phức tạp từ dữ liệu hình ảnh. ANN thuộc nhóm các thuật toán học máy giám sát, được sử dụng trong nhiều lĩnh vực như nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, dự đoán chuỗi thời gian, và nhiều ứng dụng khác.

Mạng nơ-ron nhân tạo bao gồm nhiều lớp nơ-ron (đơn vị tính toán cơ bản), trong đó mỗi nơ-ron hoạt động như một nút xử lý tín hiệu. Các thành phần chính của ANN bao gồm:

- Lớp đầu vào (Input Layer): Nhận các đặc trưng hoặc dữ liệu từ bên ngoài, mỗi nơ-ron trong lớp này đại diện cho một thuộc tính của dữ liệu đầu vào.
- Lớp ẩn (Hidden Layer): Là lớp giữa, nơi diễn ra quá trình tính toán chính. ANN có thể có một hoặc nhiều lớp ẩn, mỗi lớp bao gồm các nơ-ron. Mỗi nơ-ron nhận tín hiệu từ các nơ-ron của lớp trước đó và thực hiện một phép biến đổi phi tuyến dựa trên hàm kích hoạt.
- Lớp đầu ra (Output Layer): Đưa ra kết quả cuối cùng, có thể là các giá trị dự đoán hoặc phân loại.

Phương pháp K-Nearest Neighbors (KNN)

Phương pháp K-Nearest Neighbors (KNN) dựa trên nguyên lý so sánh khoảng cách giữa các điểm dữ liệu để thực hiện phân loại hoặc dự đoán. Khi thực hiện phân loại lá cây, KNN sẽ tìm kiếm K láng giềng gần nhất của một lá cần phân loại dựa trên các đặc trưng

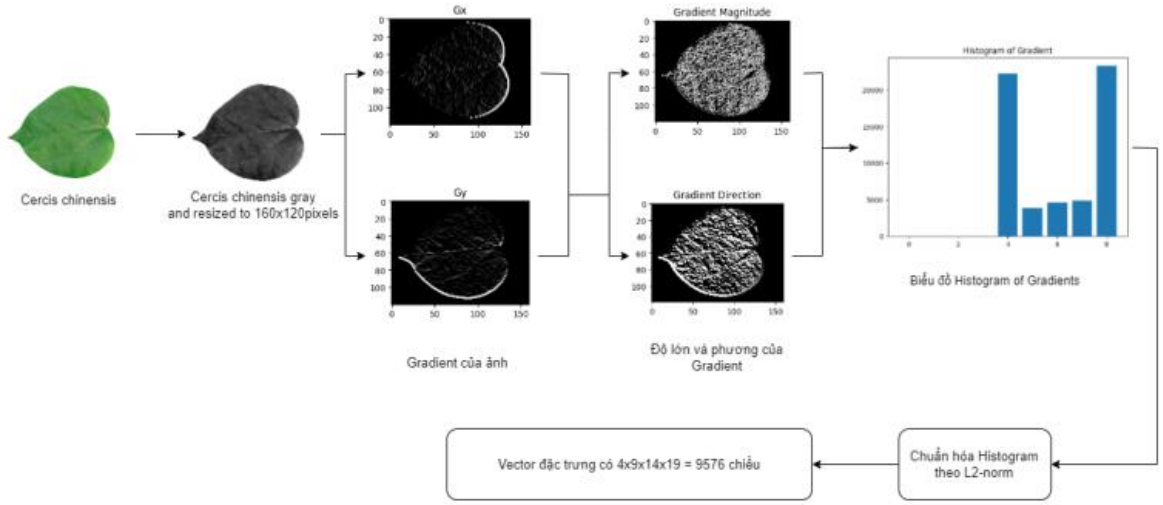
(ví dụ như Moment Hu, HOG) và sử dụng thông tin từ những láng giềng này để quyết định nhãn phân loại.

Kỹ thuật chia tập huấn luyện: Sử dụng 5-fold cross-validation để đảm bảo độ tin cậy của mô hình và hạn chế overfitting.

Hệ thống sẽ được huấn luyện và đánh giá dựa trên dữ liệu lá cây do người dùng tự thu thập, gồm các loài cây thuộc nhóm phổ biến và một số loài cá nhân, giúp mở rộng khả năng nhận dạng và phân loại của mô hình.

2.1 Đặc trưng HOG (Histogram of Oriented Gradients)

HOG dựa trên việc nhận dạng hình dạng của đối tượng cục bộ qua việc sử dụng hai ma trận quan trọng: ma trận độ lớn của gradient và ma trận hướng của gradient.



Hình 2: Sơ đồ quá trình trích xuất đặc trưng HOG

Hình 2 mô tả quá trình trích xuất đặc trưng HOG, chi tiết quá trình trích xuất đặc trưng như sau:

Bước 1: Ảnh đầu vào được chuyển sang dạng ảnh xám nhị phân và resize kích thước về 160x120pixels để giảm kích thước dữ liệu giúp đơn giản hóa quá trình xử lý và tập trung vào cấu trúc hình dạng của vật thể mà không bị ảnh hưởng bởi màu sắc. Việc thay đổi kích thước ảnh về 1 kích thước giúp thống nhất các đầu vào trong trường hợp dữ liệu chưa thống nhất.

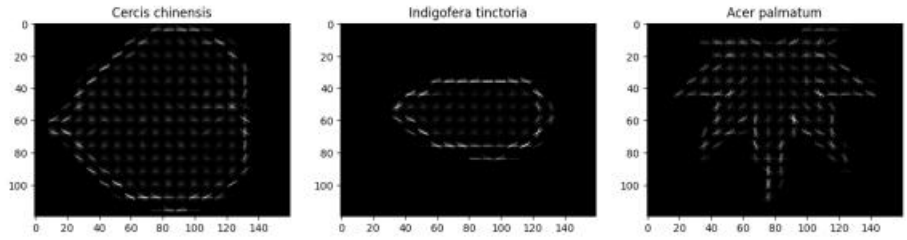
Bước 2: Tính gradient của ảnh được thực hiện bằng cách sử dụng các mặt nạ lọc, mặt nạ 3x3 được sử dụng phổ biến. Mặt nạ lọc thông dụng là mặt nạ Sobel 3x3 tính gradient theo hai hướng (x và y), và sau đó sử dụng các gradient này để tính toán các đặc trưng HOG.

Bước 3: Biểu diễn đặc trưng của ảnh thông qua 2 thông số liên quan đến mức độ thay đổi cường độ màu sắc (gradient magnitude) và hướng thay đổi cường độ màu sắc (gradient direction).

Bước 4: Tạo một bộ mô tả có thể chuyển ảnh thành vector thể hiện thông tin của gradient magnitude và gradient direction. Một biểu đồ Histogram thống kê độ lớn Gradient sẽ được tính toán trên mỗi ô cục bộ. Kích thước mỗi ô là 8x8pixels, Histogram của gradient có 9 thanh, mỗi khối là 2x2 ô và phần chồng lấn giữa các khối là 1x1 ô.

Bước 5: Chuẩn hóa Histogram của Gradient bằng chuẩn L2-Hys (sự kết hợp giữa chuẩn L2 và kỹ thuật hysteresis để đảm bảo rằng các vector đặc trưng được chuẩn hóa tốt và tránh được các vấn đề liên quan đến giá trị gradient quá nhỏ hoặc quá lớn).

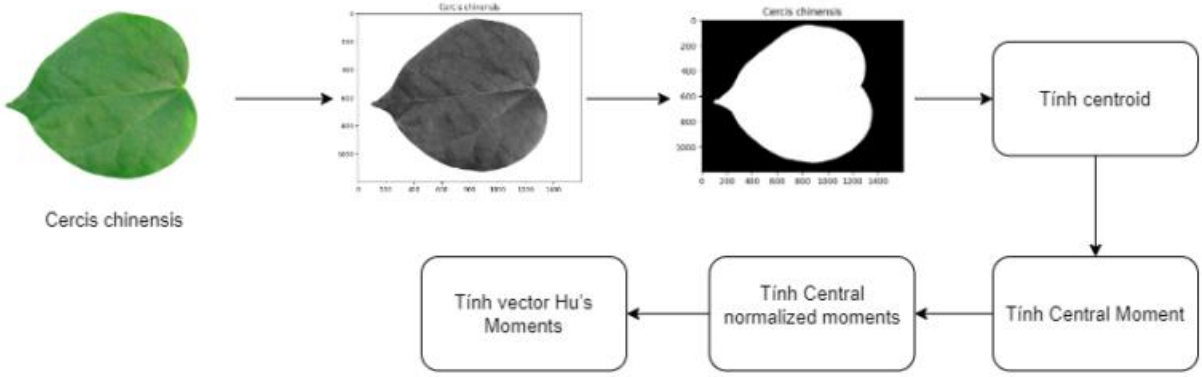
Bước 6: Xuất ra vector đặc trưng của ảnh. Trong phạm vi nghiên cứu này, ảnh đầu vào được resize thành 160x120 pixels, kích thước 1 ô là 8x8 pixels, kích thước khối là 2x2 ô, 9 giá trị/ 1 ô. Suy ra Vector đặc trưng có số chiều là [1, 9576].
Hình 3 thể hiện vector đặc trưng HOG của 1 mẫu dữ liệu trong mỗi loại lá cây



Hình 3: vector đặc trưng HOG của 3 loại lá cây

2.2 Đặc trưng HU's Moments

Hu's moments (Hu's moment invariants) là một tập hợp gồm 7 số được tính bằng cách sử dụng khoảnh khắc trung tâm (central moments) bất biến đối với các phép biến đổi hình ảnh. 6 khoảnh khắc đầu tiên đã được chứng minh là bất biến đối với chuyển động tịnh tiến, tỷ lệ, xoay và phép lật. Trong khi dấu hiệu của khoảnh khắc thứ 7 thay đổi để phản ánh hình ảnh.



Hình 4: Sơ đồ quá trình trích xuất đặc trưng HU's Moments

Để tính được HU's Moment ảnh cần được chuẩn hóa về cùng một kích thước ảnh (nghiên cứu này sử dụng kích thước 1600x1200 pixels để tính HU's Moments)
và chuyển sang dạng ảnh nhị phân với giá 0 cho nền và giá trị 1 cho đối tượng. Quá trình tính HU's Moment được mô tả như Hình

Cách tính Hu's moments

Bước 1. Tính Centroid

$$\bar{x} = \frac{\sum_x \sum_y x \cdot s(x,y)}{\sum_x \sum_y s(x,y)}, \quad \bar{y} = \frac{\sum_x \sum_y y \cdot s(x,y)}{\sum_x \sum_y s(x,y)} \tag{1}$$

Bước 2. Tính Central Moment

$$m_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q s(x,y), \quad p, q = 0, 1, 2, 3, \ldots \tag{2}$$

Bước 3. Tính Central Normalized Moments

$$M_{pq} = \frac{m_{pq}}{m_{00}^{(\frac{p+q}{2}+1)}} \tag{3}$$

Bước 4. Tính vector HU's Moments

$$S_1 = M_{20} + M_{02} \tag{4}$$

$$S_2 = (M_{20} - M_{02})^2 + 4M_{11}M_{11} \tag{5}$$

$$S_3 = (M_{30} - 3M_{12})^2 + (3M_{21} - M_{03})^2 \tag{6}$$

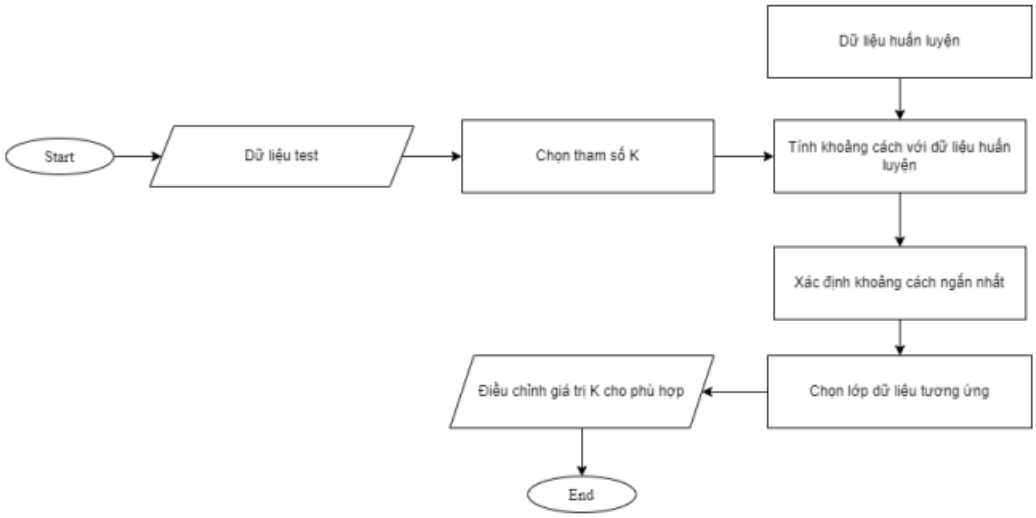
$$S_4 = (M_{30} + M_{12})^2 + (M_{21} + M_{03})^2 \tag{7}$$

$$\begin{aligned} S_5 =& (M_{30} - 3M_{12})(M_{30} + M_{12})[(M_{30} + M_{12})^2 - 3(M_{21} + M_{03})^2] \\ &+ (3M_{21} - M_{03})(M_{21} + M_{03})[3(M_{30} + M_{12})^2 - (M_{21} + M_{03})^2] \end{aligned} \tag{8}$$

$$\begin{aligned} S_6 =& (M_{20} - M_{02})[(M_{30} + M_{12})^2 - (M_{21} + M_{03})^2] \\ &+ 4M_{11}(M_{30} + M_{12})(M_{21} + M_{03}) \end{aligned} \tag{9}$$

$$\begin{aligned} S_7 =& (3M_{21} - M_{03})(M_{30} + M_{12})[(M_{30} + M_{12})^2 - 3(M_{21} + M_{03})^2] \\ &- (M_{30} - 3M_{12})(M_{21} + M_{03})[3(M_{30} + M_{12})^2 - (M_{21} + M_{03})^2] \end{aligned} \tag{10}$$

2.3 K-Nearest Neighbors (KNN)



Hình 7: Sơ đồ thuật toán KNN [6]

K-Nearest Neighbors (KNN) là một thuật toán học máy phổ biến và đơn giản, thuộc nhóm học có giám sát. Nó được sử dụng rộng rãi cho cả bài toán phân loại (classification) và hồi quy (regression). Đặc điểm nổi bật của KNN là việc không cần một mô hình cụ thể cho việc huấn luyện, mà dựa trên lưu trữ và so sánh trực tiếp với dữ liệu mẫu (dữ liệu huấn luyện). Hình 7 mô tả quá trình thực hiện của thuật toán như sau:

1. **Bước 1: Chuẩn bị dữ liệu**

Trước tiên, cần có tập dữ liệu đã được gán nhãn để huấn luyện mô hình. Tập dữ liệu này bao gồm:

- **Dữ liệu đầu vào (features):** Các đặc trưng mô tả mỗi điểm dữ liệu (ví dụ: Hu's moment, HOG,...).
- **Nhãn (labels):** Kết quả phân loại tương ứng với từng điểm dữ liệu (ví dụ: các loài lá cây).

2. **Bước 2: Xác định giá trị K**

Chọn giá trị **K** (số lượng láng giềng gần nhất) mà thuật toán sẽ sử dụng để đưa ra quyết định phân loại. Thông thường, giá trị này sẽ được thử nghiệm với nhiều giá trị khác nhau để tìm ra số K tối ưu cho hiệu quả phân loại cao nhất.

3. **Bước 3: Tính khoảng cách Manhattan**

Khi cần phân loại một điểm dữ liệu mới, thuật toán KNN với **Manhattan distance** sẽ tính toán khoảng cách giữa điểm mới này và tất cả các điểm trong tập dữ liệu huấn luyện. Khoảng cách Manhattan giữa hai điểm dữ liệu $(x_1, x_2, ..., x_n)$ và $(y_1, y_2, ..., y_n)$ được tính theo công thức:

$$\text{Manhattan Distance} = \sum_{i=1}^{n} |x_i - y_i|$$

Trong đó:

- x_i và y_i là các giá trị của đặc trưng thứ i của hai điểm dữ liệu.
- n là số lượng đặc trưng.

4. **Bước 4: Xác định K láng giềng gần nhất**

Sau khi tính toán khoảng cách Manhattan giữa điểm mới và tất cả các điểm trong tập dữ liệu huấn luyện, thuật toán sẽ chọn ra **K điểm** có khoảng cách nhỏ nhất (gần nhất) với điểm mới này. Những điểm này được gọi là **K láng giềng gần nhất**.

5. **Bước 5: Phân loại dựa trên K láng giềng**

- **Phân loại:** Với các điểm K láng giềng đã xác định, thuật toán sẽ kiểm tra nhãn của các điểm đó. KNN thực hiện phân loại bằng cách **bỏ phiếu đa số** (majority voting): nhãn xuất hiện nhiều nhất trong số K láng giềng sẽ được gán cho điểm dữ liệu mới.

  Ví dụ, nếu trong K láng giềng gần nhất có 3 nhãn thuộc lớp A và 2 nhãn thuộc lớp B, thì điểm mới sẽ được phân loại vào lớp A.

- **Hồi quy:** Nếu KNN được sử dụng cho hồi quy, giá trị đầu ra sẽ là trung bình hoặc giá trị trung bình có trọng số của các giá trị đầu ra của K láng giềng gần nhất.

6. **Bước 6: Đánh giá mô hình**

Khi hoàn thành phân loại cho các điểm dữ liệu mới, kết quả có thể được đánh giá bằng cách so sánh nhãn dự đoán với nhãn thực tế, sử dụng các chỉ số như độ chính xác (accuracy), độ nhạy (sensitivity), độ đặc hiệu (specificity), hoặc F1-score.

Ngoài ra, để đảm bảo mô hình không bị overfitting hoặc underfitting, cần thực hiện đánh giá mô hình bằng các phương pháp như **cross-validation** (ví dụ: 5-fold cross-validation) và thử nghiệm với các giá trị K khác nhau.

CHƯƠNG 2: THỰC HIỆN

Mục tiêu của bài tập là xây dựng một hệ thống tự động phân loại lá cây sử dụng trích phương pháp trích xuất đặc trưng Hu's moments và HOG, kết hợp với phương pháp KNN và ANN.

Bước 1: Chuẩn bị dữ liệu.

Mô tả dữ diệu đầu vào gồm có 5 lớp, mỗi lớp có 50 mẫu, đã được xử lý để loại bỏ bóng, véc tơ. Dữ liệu bao gồm lá cây tử đẳng(la chi), lá phong, lá táo, rau muóng, lá chàm.

Dữ liệu lá rau muống và lá táo được mượn của hai bạn Ngô Hữu Minh và Nguyễn Thành Lan. Trân trọng cảm ơn hai bạn.

Data/leaf

Cấu trúc dữ liệu như sau:

- Số lượng mẫu dữ liệu (samples): 250 ảnh JPG. Trong đó 150 ảnh (lá cây tử đẳng, lá phong, lá chàm) có kích thước 1600x1200. Và 100 ảnh rau muống có kích thước 4096x3072 và lá táo có kích thước 4032x3024

Bước 2: Trích đặc trưng Hu's moment và đặc trưng HOG.

Trích xuất đặc trưng HOG

```
def extract_hog_features(image):
    # Trích xuất đặc trưng HOG từ ảnh xám
    features = hog(image, orientations=9, pixels_per_cell=(8, 8),
                   cells_per_block=(2, 2), visualize=False)
    return features

def extract_hog_features_from_folder(path, label, size=(128, 64),
output_gray_folder='gray_images'):  # Thêm tham số cho thư mục lưu ảnh xám
    list_of_files = os.listdir(path)
    features = []
    labels = []

    # Tạo thư mục lưu ảnh xám nếu chưa tồn tại
    os.makedirs(output_gray_folder, exist_ok=True)
```

```
    for i in list_of_files:
        img = plt.imread(os.path.join(path, i))  # Đọc ảnh từ đường dẫn
        if img.ndim == 3:  # Nếu ảnh là ảnh màu, chuyển sang không gian màu HSV
            img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
            img_gray = img_hsv[:, :, 2]  # Chọn kênh V (Value) từ ảnh HSV
        else:
            img_gray = img  # Nếu ảnh đã là ảnh xám

        # Lưu ảnh xám
        gray_image_path = os.path.join(output_gray_folder, f'gray_{i}')
        cv2.imwrite(gray_image_path, img_gray)
        # Thay đổi kích thước ảnh
        img_gray = cv2.resize(img_gray, size)  #

        hog_features = extract_hog_features(img_gray)  # Trích xuất đặc trưng HOG
        features.append(hog_features)
        labels.append(label)
    return features, labels

def save_to_csv(features, labels, file_name):
    df = pd.DataFrame(features)
    df['label'] = labels
    df.to_csv(file_name, index=False)

def standardize_features(input_csv, output_csv):
    # Tải tệp CSV vào DataFrame
    df = pd.read_csv(input_csv)

    # Áp dụng chuẩn hóa Z-score cho các cột đặc trưng (giả sử cột cuối cùng là nhãn)
    feature_columns = df.columns[:-1]  # Loại bỏ cột nhãn
    df[feature_columns] = df[feature_columns].apply(zscore)

    # Lưu dữ liệu đã chuẩn hóa vào tệp CSV mới
    df.to_csv(output_csv, index=False)

# Đường dẫn ảnh đầu vào và lưu lại ảnh xám (DATA LEAF)
output_gray_folder = r'E:/Downloads/DATA/Hog/gray_images'  # Thư mục lưu ảnh xám
la_chi, nhanlachi =
extract_hog_features_from_folder(      r"E:/Downloads/DATA/LEAF/THAIDUCTOAN/
la_chi", 1, output_gray_folder=output_gray_folder)              #lachi
la_cham, nhanlacham =
extract_hog_features_from_folder(    r"E:/Downloads/DATA/LEAF/TRANTHANHKH
OA/la_cham", 2, output_gray_folder=output_gray_folder)    #lacham
```

```
la_phong, nhanlaphong =
extract_hog_features_from_folder(   r"E:/Downloads/DATA/LEAF/MAIDUCKHIEM/la
_phong", 3, output_gray_folder=output_gray_folder)                #laphong
la_tao, nhanlatao =
extract_hog_features_from_folder(       r"E:/Downloads/DATA/LEAF/NGOHUUMINH/l
a tao", 4, output_gray_folder=output_gray_folder)                                #la tao
rau_muong, nhanraumuong = extract_hog_features_from_folder(
r"E:/Downloads/DATA/LEAF/NGUYENTHANHLAN/rau muong", 5,
output_gray_folder=output_gray_folder)                          #rau muong

# Lưu vào file csv
features = la_chi + la_cham + la_phong + la_tao + rau_muong
labels = nhanlachi + nhanlacham + nhanlaphong + nhanlatao + nhanraumuong
save_to_csv(features, labels, r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11.csv')

# Chuẩn hóa dữ liệu HOG
input_csv_path =  r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11.csv'
output_csv_path = r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11Std.csv'
standardize_features(input_csv_path, output_csv_path)
```

1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
3	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
4	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
5	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
6	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
7	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
8	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
9	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
10	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
11	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
12	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
13	4.255102	-0.68677	-0.70584	-0.69618	4.553534	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
14	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
15	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
16	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
17	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
18	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
19	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
20	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
21	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
22	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
23	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
24	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
25	2.786303	-0.68677	-0.70584	-0.69618	3.005566	-0.70219	-0.70592	-0.65619	-0.59219	3.444653	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
26	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
27	4.255102	-0.68677	-0.70584	-0.69618	4.553534	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
28	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763
29	-0.75969	-0.68677	-0.70584	-0.69618	-0.73156	-0.70219	-0.70592	-0.65619	-0.59219	-0.81581	-0.70643	-0.72876	-0.73365	-0.73151	-0.72763

0.762981	1.059843	1.27237	1.215897	0.183716	1.380603	1.373327	0.446998	0.379136	1.273023	1.944454		5
2.094295	1.199484	1.32508	2.322485	2.223055	0.004683	1.211522	1.092801	-0.37424	0.828585	1.47037		5
0.680652	1.801863	1.323878	1.479318	0.512843	2.124769	1.427194	1.381963	1.140794	-0.28703	1.956548		5
1.255878	1.312517	1.179235	2.073269	2.010013	0.89475	1.275928	0.763968	1.93463	0.90603	0.018348		5
2.100806	-0.21614	1.275922	1.272403	0.843494	0.192187	0.727073	1.057008	0.355524	0.772684	2.211628		5
1.952728	1.741438	0.60367	1.965852	2.003147	-0.7195	1.262785	1.178507	2.503254	1.940831	0.414718		5
1.355397	-0.04574	1.343901	0.738944	1.310374	-0.11528	1.16517	1.254158	1.446803	1.71404	0.533634		5
2.083849	1.058973	0.977938	0.268498	0.49391	0.465586	1.363958	1.360916	2.334817	1.131672	1.629339		5
1.75091	1.292158	1.190518	-0.16543	0.282317	1.790178	1.287727	0.80545	2.558122	1.925726	0.258725		5
1.969635	0.721737	1.255819	2.225504	0.759825	0.366257	1.064871	-0.5335	2.663905	-0.07836	1.522594		5
2.137334	1.343263	1.302873	1.078113	0.099361	1.274886	1.266461	1.924806	0.571489	1.78564	1.575446		5
2.108123	0.802414	1.281321	1.895678	0.500711	0.20871	0.502987	-0.07544	0.524542	2.689054	0.812077		5
1.869978	0.5497	1.268027	0.050608	2.143854	1.072028	1.368786	2.149182	0.844784	-0.68139	2.220181		5
2.108982	-0.63795	1.281955	1.22236	0.895601	1.374323	1.383351	0.154606	0.792667	-0.68139	0.497228		5
1.4621	0.321834	1.232314	0.553094	0.073175	0.885915	1.297472	0.558984	1.535959	1.484218	1.546495		5
0.225458	1.895176	1.253781	0.411485	-0.24231	1.235993	1.353887	1.071569	1.537582	0.598538	2.361517		5
1.06231	2.755873	1.227475	-0.01174	1.04568	1.887723	0.533122	2.280393	-0.34134	0.234392	2.640479		5
2.027475	1.575644	1.221818	2.177894	1.146608	1.179363	0.89751	0.224814	2.608825	1.221402	0.345384		5
1.967842	0.14622	1.177819	2.116285	2.018627	1.596963	0.411695	1.4235	-0.19447	1.883114	0.618035		5

Trích xuất đặc trưng Hu’s Moments (7 features)

```
def extract_feature(image):
    # Trích xuất đặc trưng Hu Moments từ ảnh
```

```
    moments = cv2.moments(image)
    hu_moments = cv2.HuMoments(moments)
    return hu_moments.flatten()


def extract_hu_features_from_folder(path, label, output_binary_folder='binary_images'):
    list_of_files = os.listdir(path)
    features = []
    labels = []

    # Tạo thư mục lưu ảnh nhị phân nếu chưa tồn tại
    os.makedirs(output_binary_folder, exist_ok=True)

    for i in list_of_files:
        img = plt.imread(os.path.join(path, i))  # Đọc ảnh từ đường dẫn
        if img.ndim == 3:  # Nếu ảnh là ảnh màu, chuyển sang không gian màu xám
            img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        else:
            img_gray = img

        # Chuyển ảnh xám sang nhị phân bằng ngưỡng Otsu
        _, img_binary = cv2.threshold(img_gray, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

        # Lưu ảnh nhị phân
        binary_image_path = os.path.join(output_binary_folder, f'binary_{i}')
        cv2.imwrite(binary_image_path, img_binary)

        # Trích xuất đặc trưng Hu Moments từ ảnh nhị phân
        hu_features = extract_feature(img_binary)
        features.append(hu_features)
        labels.append(label)

    return features, labels

def save_to_csv(features, labels, file_name):
    # Lưu các đặc trưng và nhãn vào tệp CSV
    df = pd.DataFrame(features)
    df['label'] = labels
    df.to_csv(file_name, index=False)

def z_score_standardization(input_csv, output_csv):
    # Tải tệp CSV vào DataFrame
```

```
    df = pd.read_csv(input_csv)

    # Áp dụng chuẩn hóa Z-score cho các cột đặc trưng (giả sử cột cuối cùng là nhãn)
    feature_columns = df.columns[:-1]  # Loại bỏ cột nhãn
    df[feature_columns] = df[feature_columns].apply(zscore)

    # Lưu dữ liệu đã chuẩn hóa vào tệp CSV mới
    df.to_csv(output_csv, index=False)

output_binary_folder = r'D:/DATA/Hu/binary_images'  # Thư mục lưu ảnh nhị phân
la_chi, nhanlachi =
extract_hu_features_from_folder(      r"D:/DATA/LEAF/THAIDUCTOAN/la_tu_dang",
1, output_binary_folder=output_binary_folder)  # la chi
la_cham, nhanlacham =
extract_hu_features_from_folder(     r"D:/DATA/LEAF/TRANTHANHKHOA/la_cham"
, 2, output_binary_folder=output_binary_folder)  # la cham
la_phong, nhanlaphong =
extract_hu_features_from_folder(   r"D:/DATA/LEAF/MAIDUCKHIEM/la_phong", 3,
output_binary_folder=output_binary_folder)  # la phong
la_tao, nhanlatao =
extract_hu_features_from_folder(      r"D:/DATA/LEAF/NGOHUUMINH/la tao", 4,
output_binary_folder=output_binary_folder)  # la tao
rau_muong, nhanraumuong = extract_hu_features_from_folder(
r"D://DATA/LEAF/NGUYENTHANHLAN/rau muong", 5,
output_binary_folder=output_binary_folder)  # rau muong

# Lưu vào file csv
features = la_chi + la_cham + la_phong + la_tao + rau_muong
labels = nhanlachi + nhanlacham + nhanlaphong + nhanlatao + nhanraumuong
save_to_csv(features, labels, r'D:/DATA/Hu/hutest_hsv/HUnhom11.csv')

# Chuẩn hóa dữ liệu Hu Moments
input_csv_path =    r'D:/DATA/Hu/hutest_hsv/HUnhom11.csv'
output_csv_path =   r'D:/DATA/Hu/hutest_hsv/HUnhom11Std.csv'
z_score_standardization(input_csv_path, output_csv_path)
```

Bước 3: Thực hiện phân loại lá cây dùng phương pháp KNN. Thay đổi giá trị K để thấy ảnh hưởng của K đến hiệu quả của hệ thống. Đánh giá hệ thống dùng phương pháp 5-fold cross validation.

	A	B	C	D	E	F	G	H
1	0	1	2	3	4	5	6	label
2	2.450311	2.671338	0.648252	-0.23543	-0.13102	0.011396	0.040067	1
3	1.233603	1.204877	-0.18973	-0.0438	-0.12938	0.067318	0.054482	1
4	2.278347	2.090805	-0.01879	0.305332	-0.07901	0.143254	0.044575	1
5	1.625582	1.743177	-0.05395	0.304022	-0.0832	0.364988	0.032723	1
6	2.487851	2.07085	0.592086	0.265434	-0.04264	0.568895	0.064643	1
7	1.885435	2.038204	-0.22538	0.597926	-0.07818	0.953854	0.072108	1
8	1.65873	1.601432	3.470503	2.878021	1.966317	3.159225	-0.8036	1
9	1.539463	1.550694	0.020904	-0.38396	-0.14635	-0.19245	0.047544	1
10	2.738285	2.676615	3.836065	8.01002	6.612645	7.827894	-11.6094	1
11	2.308701	2.46419	0.501651	-0.27833	-0.15651	-0.36145	0.057144	1
12	1.584193	1.762397	0.606101	0.009684	-0.10311	0.173958	0.102971	1
13	2.285573	2.680885	-0.08033	0.760742	-0.03516	1.248283	0.156314	1
14	1.547902	1.361473	-0.07246	0.364505	-0.18914	-0.88687	0.152122	1
15	2.312939	2.343254	0.788863	0.015318	-0.14064	-0.09154	0.156955	1
16	1.553475	1.690747	0.306512	-0.17991	-0.16441	-0.45022	0.064343	1
17	1.370087	1.18229	-0.3447	-0.32246	-0.14732	-0.14181	0.047557	1
18	2.998373	3.124126	0.497779	-0.25197	-0.15311	-0.23578	0.069057	1
19	1.970227	2.081938	0.179566	0.171036	-0.17631	-0.84509	-0.06045	1
20	1.645789	1.874762	0.000322	-0.32635	-0.15082	-0.29006	0.050394	1
21	1.388035	1.195921	-0.31969	-0.27433	-0.14625	-0.12401	0.049838	1
22	1.070457	0.906943	0.298083	0.134737	-0.10391	0.070052	0.134283	1
23	0.697935	0.427965	-0.27062	-0.19226	-0.14095	-0.09936	0.043552	1

88	-0.50549	-0.58227	-0.34626	-0.30769	-0.14681	-0.20254	0.044566	2
89	-0.57367	-0.58014	-0.34351	-0.33015	-0.14727	-0.22767	0.044451	2
90	-0.44932	-0.55077	-0.34529	-0.3321	-0.14715	-0.21594	0.044667	2
91	-0.37353	-0.5165	-0.34254	-0.32369	-0.14694	-0.21462	0.044542	2
92	-0.62198	-0.60696	-0.35089	-0.36724	-0.14722	-0.2229	0.045584	2
93	-0.49416	-0.57722	-0.35203	-0.35542	-0.14718	-0.21758	0.045497	2
94	-0.46508	-0.54526	-0.35511	-0.40381	-0.14741	-0.23326	0.045971	2
95	-0.4464	-0.59156	-0.34087	-0.18755	-0.14456	-0.14724	0.04375	2
96	-0.47077	-0.5566	-0.35291	-0.3754	-0.14727	-0.22227	0.046121	2
97	-0.46525	-0.54279	-0.34708	-0.35708	-0.14723	-0.22681	0.045188	2
98	-0.47008	-0.54858	-0.3556	-0.40635	-0.14742	-0.23419	0.045955	2
99	-0.52551	-0.55258	-0.35652	-0.42164	-0.14747	-0.24463	0.045977	2
100	-0.61353	-0.57869	-0.35693	-0.42442	-0.14748	-0.24789	0.04597	2
101	-0.42728	-0.49687	-0.3385	-0.33718	-0.14768	-0.25419	0.047491	2
102	0.027216	-0.02959	-0.10366	-0.08221	-0.16708	-0.48239	0.043231	3
103	-0.06848	0.05101	0.222297	-0.00647	-0.18016	-0.50014	0.004266	3
104	-0.08161	-0.04319	0.012515	-0.10626	-0.1643	-0.42725	0.02143	3
105	-0.23876	-0.0898	0.116418	-0.09222	-0.16695	-0.42069	0.014681	3
106	-0.03984	-0.04377	-0.06515	-0.08553	-0.16819	-0.47827	0.049276	3
107	-0.1444	-0.07772	-0.03022	-0.17917	-0.15947	-0.39716	0.058848	3
108	-0.25594	-0.18969	-0.06341	-0.15626	-0.15279	-0.32324	0.019954	3
109	-0.24724	-0.0539	0.205786	-0.09863	-0.1691	-0.43026	0.015084	3
110	-0.25204	-0.1975	-0.1421	-0.23219	-0.15543	-0.37092	0.047009	3
111	-0.2466	-0.2361	0.026152	0.097392	-0.13522	-0.18891	-0.03398	3

KNN-HOG:

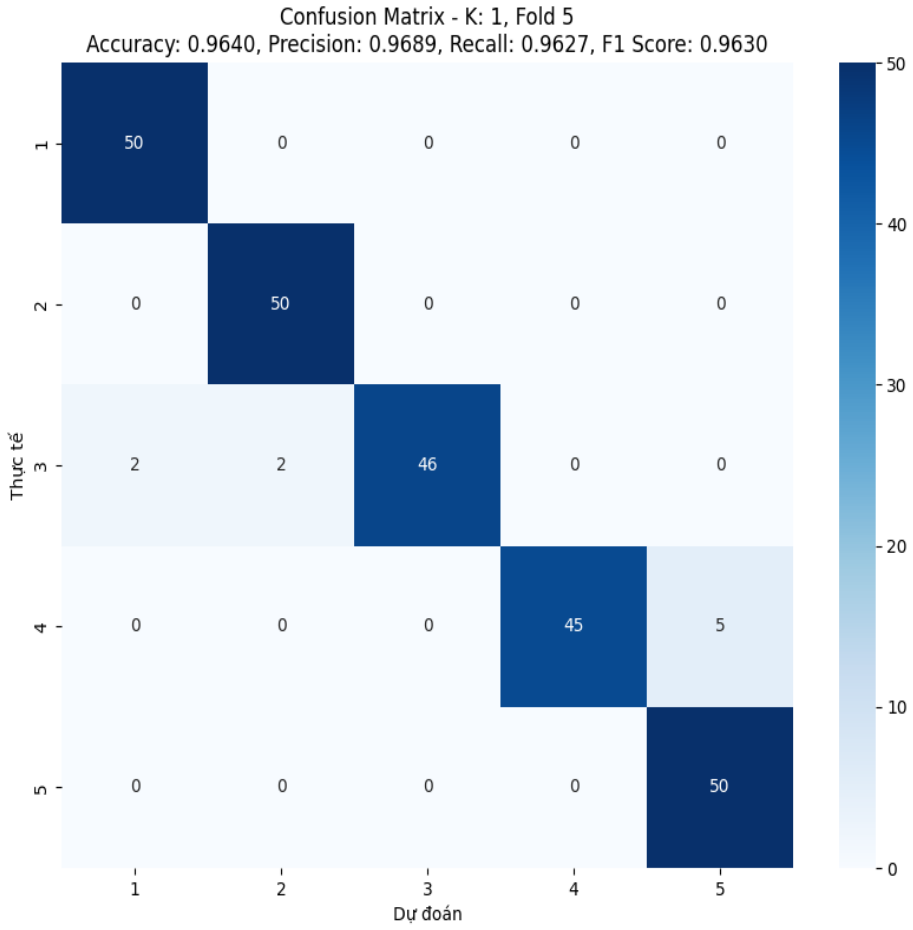
```
# Thiết lập mã hóa
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')


# Bước 1: Tải dataset từ file CSV
dataset = pd.read_csv(r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11Std.csv')

# Hiển thị số mẫu dữ liệu tương ứng với từng nhãn
print(dataset.groupby('label').size())

# Bước 2: Chia dữ liệu và nhãn
X = dataset.drop('label', axis=1)
y = dataset['label'].astype(str)  # Chuyển đổi y thành chuỗi

# Xác định tất cả các nhãn có trong dataset và chuyển đổi thành chuỗi
```

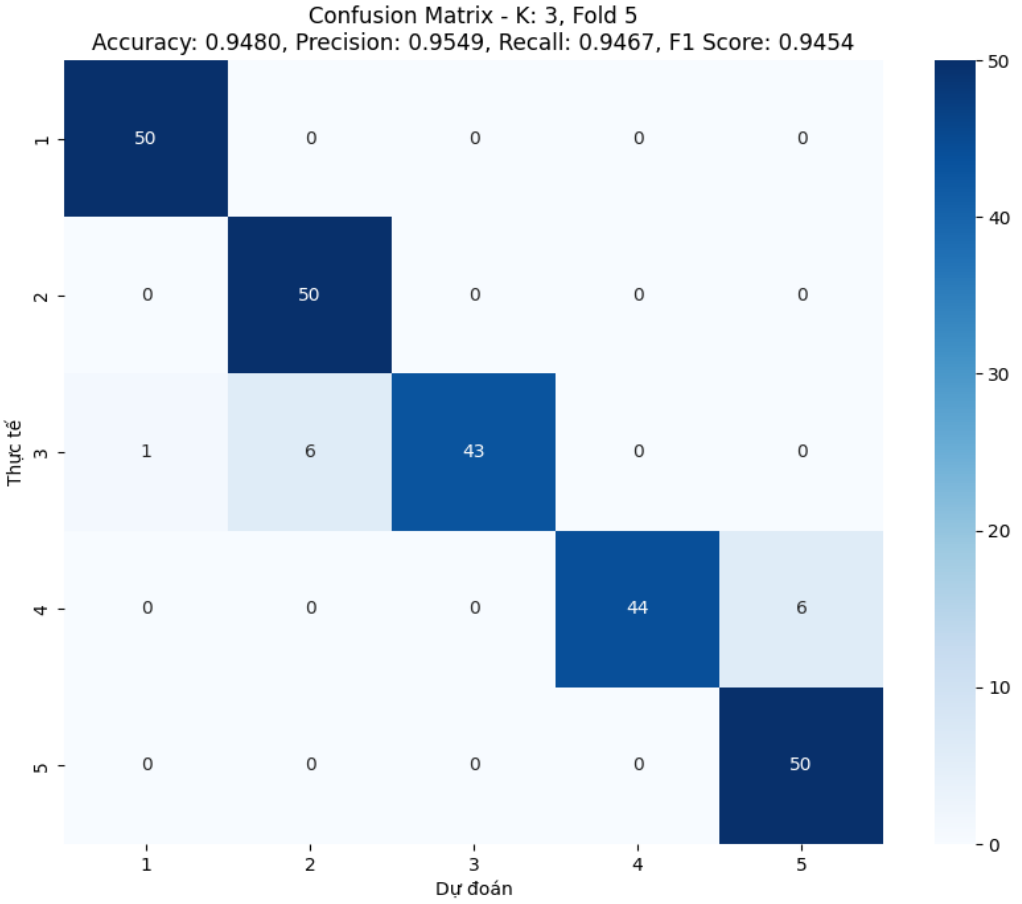
```
all_labels = sorted(y.unique())  # Đảm bảo các nhãn đều là chuỗi
num_classes = len(all_labels)

# Khởi tạo k-fold cross-validation
kf = KFold(n_splits=5, random_state=42, shuffle=True)

# Danh sách các giá trị K để thử nghiệm
k_values = [1, 3, 5, 7]

# Biến lưu kết quả
results = []

# Bước 3 vòng lặp qua các giá trị K
for k in k_values:
    print(f"\nThử nghiệm với K = {k}")

    # Khởi tạo mô hình KNN với K hàng xóm
    classifier = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)

    # Khởi tạo biến lưu trữ kết quả trung bình cho các chỉ số hiệu suất
    accuracy_tb = precision_tb = recall_tb = f1_tb = 0
    cm = np.zeros((num_classes, num_classes))  # Ma trận nhầm lẫn tổng có kích thước cố định

    fold = 0
    for train_index, test_index in kf.split(X):
        fold += 1
        # Chia dữ liệu cho từng fold
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Huấn luyện mô hình KNN
        classifier.fit(X_train, y_train)

        # Dự đoán trên tập kiểm thử
        y_pred = classifier.predict(X_test)

        # Hiển thị báo cáo phân loại cho từng fold
        print(f"Results for fold {fold} with K = {k}:")
        print(classification_report(y_test, y_pred, target_names=all_labels))

        # Ma trận nhầm lẫn cho từng fold
        cm_fold = confusion_matrix(y_test, y_pred, labels=all_labels)
```

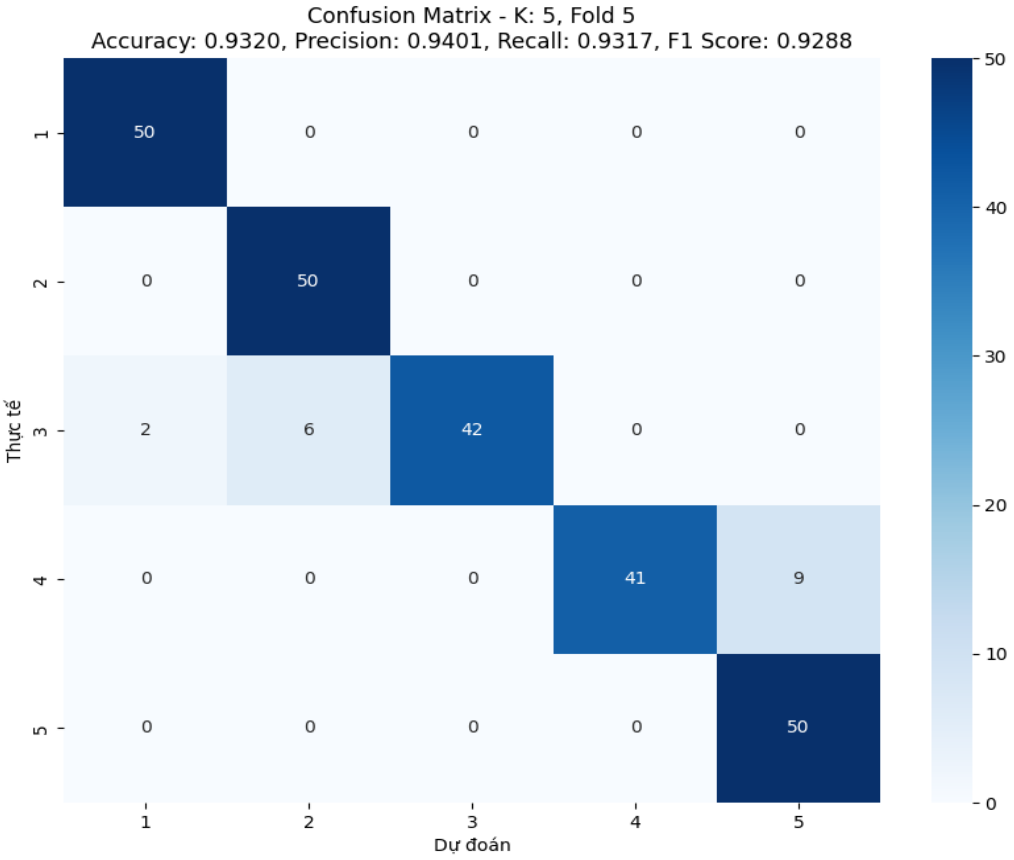
```
    cm += cm_fold  # Cộng dồn ma trận nhầm lẫn, đảm bảo kích thước cố định

    # Tính toán các chỉ số hiệu suất
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')

    # Hiển thị các chỉ số cho từng fold
    print(f"Accuracy for fold {fold}: {accuracy:.4f}")
    print(f"Precision for fold {fold}: {precision:.4f}")
    print(f"Recall for fold {fold}: {recall:.4f}")
    print(f"F1 score for fold {fold}: {f1:.4f}")
    print("-------------------------------------------------")

    # Cộng dồn các chỉ số để tính trung bình sau tất cả các fold
    accuracy_tb += accuracy
    precision_tb += precision
    recall_tb += recall
    f1_tb += f1

# Tính và hiển thị các chỉ số hiệu suất trung bình cho tất cả các fold
accuracy_tb /= fold
precision_tb /= fold
recall_tb /= fold
f1_tb /= fold

# Lưu kết quả cho giá trị K hiện tại
results.append({
    'K': k,
    'fold': fold,
    'accuracy': accuracy_tb,
    'precision': precision_tb,
    'recall': recall_tb,
    'f1': f1_tb,
    'confusion_matrix': cm  # Lưu ma trận nhầm lẫn tổng
})

print(f"K = {k} - Accuracy average: {accuracy_tb:.4f}")
print(f"K = {k} - Precision average: {precision_tb:.4f}")
print(f"K = {k} - Recall average: {recall_tb:.4f}")
print(f"K = {k} - F1 score average: {f1_tb:.4f}")
```

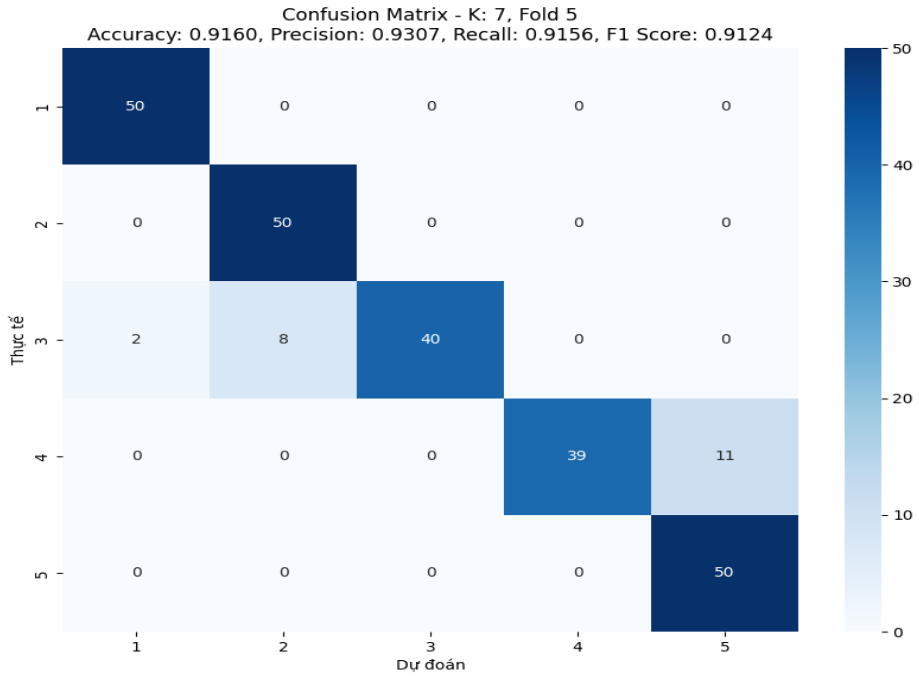
```
    # Vẽ ma trận nhầm lẫn
    class_labels = [str(label) for label in all_labels]  # Nhãn lớp
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='g', xticklabels=class_labels,
yticklabels=class_labels, cmap='Blues')
    plt.title(f'Confusion Matrix - K: {k}, Fold {fold}\nAccuracy: {accuracy_tb:.4f},
Precision: {precision_tb:.4f}, Recall: {recall_tb:.4f}, F1 Score: {f1_tb:.4f}')
    plt.xlabel('Dự đoán')
    plt.ylabel('Thực tế')
    plt.savefig(f'confusion_matrix_k_{k}_fold_{fold + 1}.png')
    plt.close()  # Đóng biểu đồ để tránh hiện thị đè lên nhau

# Hiển thị kết quả tổng quan cho tất cả các giá trị K đã thử nghiệm
print("\nTổng quan kết quả:")
for result in results:
    print(f"K = {result['K']} - Accuracy: {result['accuracy']:.4f}, Precision:
{result['precision']:.4f}, Recall: {result['recall']:.4f}, F1 Score: {result['f1']:.4f}")

# Lưu kết quả vào file CSV
results_df = pd.DataFrame(results)
results_df.to_csv('results_KNN_HOG.csv', index=False)
```

Confusion matrix


Confusion Matrix - K: 1, Fold 5
Accuracy: 0.9640, Precision: 0.9689, Recall: 0.9627, F1 Score: 0.9630
Thực tế
Dự đoán
1
2
3
4
5
50
0
0
0
0
0
50
0
0
0
2
2
46
0
0
0
0
0
45
5
0
0
0
0
50
50
40
30
20
10
0


Confusion Matrix - K: 3, Fold 5
Accuracy: 0.9480, Precision: 0.9549, Recall: 0.9467, F1 Score: 0.9454
Thực tế
Dự đoán
1
2
3
4
5
50
0
0
0
0
0
50
0
0
0
1
6
43
0
0
0
0
0
44
6
0
0
0
0
50
50
40
30
20
10
0


Confusion Matrix - K: 5, Fold 5
Accuracy: 0.9320, Precision: 0.9401, Recall: 0.9317, F1 Score: 0.9288
Thực tế
Dự đoán
1 2 3 4 5
50 0 0 0 0
0 50 0 0 0
2 6 42 0 0
0 0 0 41 9
0 0 0 0 50
0 10 20 30 40 50


Confusion Matrix - K: 7, Fold 5
Accuracy: 0.9160, Precision: 0.9307, Recall: 0.9156, F1 Score: 0.9124
Thực tế
Dự đoán
1 2 3 4 5
50 0 0 0 0
0 50 0 0 0
2 8 40 0 0
0 0 0 39 11
0 0 0 0 50
0 10 20 30 40 50

Kết quả được lưu vào file csv tổng quát quát.

	A	B	C	D	E	F	G
1	K	fold	accuracy	precision	recall	f1	confusion_
2	1	5	0.964	0.968936	0.96268	0.963023	[[50. 0. 0.
3	3	5	0.948	0.954874	0.946662	0.945374	[[50. 0. 0.
4	5	5	0.932	0.940102	0.931693	0.928808	[[50. 0. 0.
5	7	5	0.916	0.930735	0.915612	0.912381	[[50. 0. 0.

đánh giá về ảnh hưởng của số lượng K đến hiệu quả của hệ thống phân loại lá cây sử dụng phương pháp KNN dựa trên đặc trưng HOG, đánh giá qua phương pháp 5-fold cross-validation.

1. Accuracy:

- Khi K=1, độ chính xác cao nhất đạt 0.964, cho thấy rằng hệ thống phân loại lá cây hoạt động tốt nhất khi chỉ dựa vào một láng giềng gần nhất.
- Khi K tăng lên (3, 5, 7), độ chính xác giảm nhẹ, xuống còn 0.948 ở K= 3, 0.932 ở K=5, và 0.916 ở K=7.
- Sự giảm dần trong độ chính xác khi K tăng phản ánh rằng mô hình có xu hướng trở nên ít chính xác hơn khi phải xem xét nhiều láng giềng hơn.

2. Precision:

- Precision giảm từ 0.969 khi K=1 xuống 0.931 khi K=7.
- Điều này cho thấy mô hình tăng khả năng dự đoán nhầm khi K tăng, tức là việc phân loại có thể không còn chính xác cho từng lớp khi nhiều láng giềng được đưa vào tính toán.

3. Recall:

- Recall cũng giảm từ 0.963 khi K=1 xuống 0.916 khi K=7.
- Điều này chỉ ra rằng hệ thống bỏ sót một số mẫu của lớp thực khi K tăng lên, do đó, giảm hiệu quả trong việc phân loại chính xác các mẫu dương tính thực sự.

4. F1 Score:

- F1 Score giảm từ 0.963 ở K=1 xuống 0.912 ở K=7.
- Sự sụt giảm F1 Score phản ánh sự giảm hiệu suất tổng thể khi cân bằng giữa precision và recall, cho thấy sự phân loại trở nên kém hiệu quả hơn khi K lớn.

5. Confusion Matrix:

- Khi K=1, ma trận nhầm lẫn có rất ít lỗi (chỉ một vài mẫu bị nhầm giữa các lớp như lớp 3 và lớp 4).
- Khi K tăng, số lượng nhầm lẫn giữa các lớp tăng nhẹ, đặc biệt là giữa các lớp gần nhau. Ví dụ, với K=7, lớp 3 và lớp 4 có số lượng nhầm lẫn tăng, cho thấy rằng các lớp dễ bị lẫn lộn hơn khi xem xét nhiều láng giềng.

Kết luận:

- Khi tăng K, hiệu quả tổng thể của hệ thống phân loại dựa trên HOG và KNN giảm dần, đặc biệt về độ chính xác, precision, recall và F1 score.
- Mô hình KNN-HOG đạt hiệu quả cao nhất khi K=1, cho thấy việc dựa vào láng giềng gần nhất mang lại độ chính xác cao nhất trong trường hợp này.
- Với các đặc trưng HOG, mô hình hoạt động tốt hơn ở giá trị K thấp, vì đặc trưng HOG có thể phân biệt tốt các lớp với một láng giềng gần nhất hơn là với nhiều láng giềng.

KNN-HU:

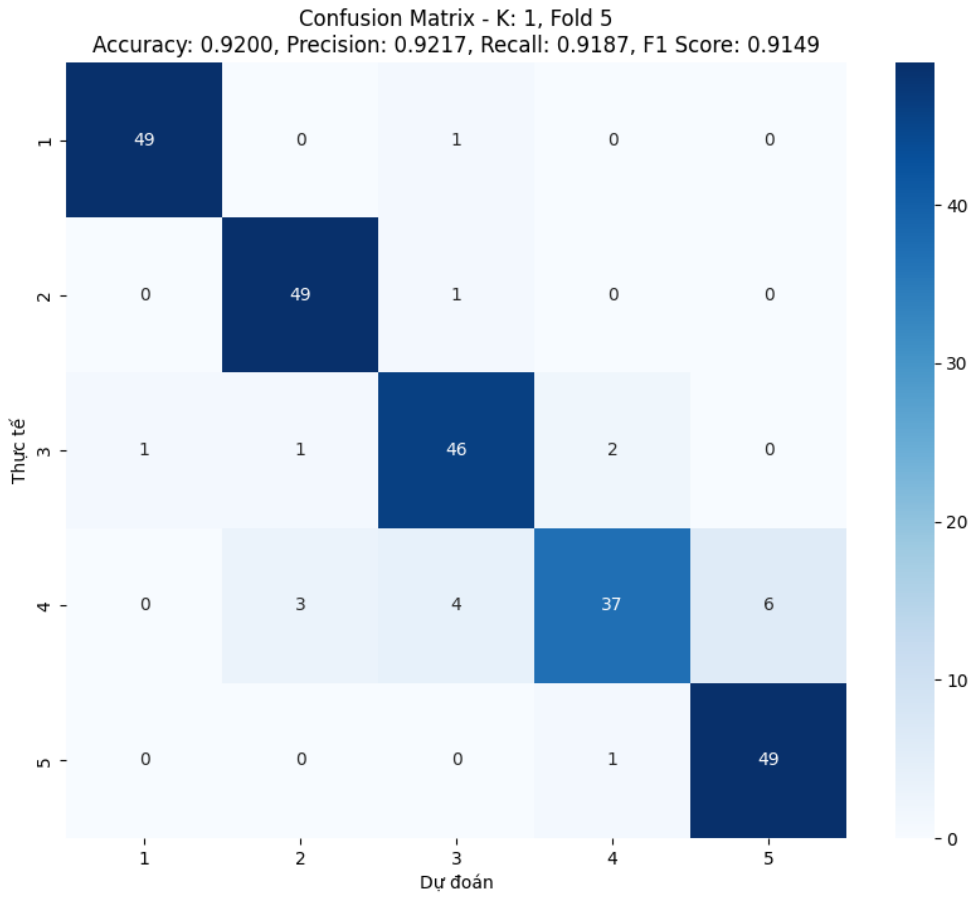
```
# Thiết lập mã hóa
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')


# Bước 1: Tải dataset từ file CSV
dataset = pd.read_csv(r'D:/DATA/Hu/hutest_hsv/HUnhom11Std.csv'
)

# Hiển thị số mẫu dữ liệu tương ứng với từng nhãn
print(dataset.groupby('label').size())

# Bước 2: Chia dữ liệu và nhãn
X = dataset.drop('label', axis=1)
y = dataset['label'].astype(str)  # Chuyển đổi y thành chuỗi

# Xác định tất cả các nhãn có trong dataset và chuyển đổi thành chuỗi
all_labels = sorted(y.unique())  # Đảm bảo các nhãn đều là chuỗi
num_classes = len(all_labels)

# Khởi tạo k-fold cross-validation
kf = KFold(n_splits=5, random_state=42, shuffle=True)

# Danh sách các giá trị K để thử nghiệm
```

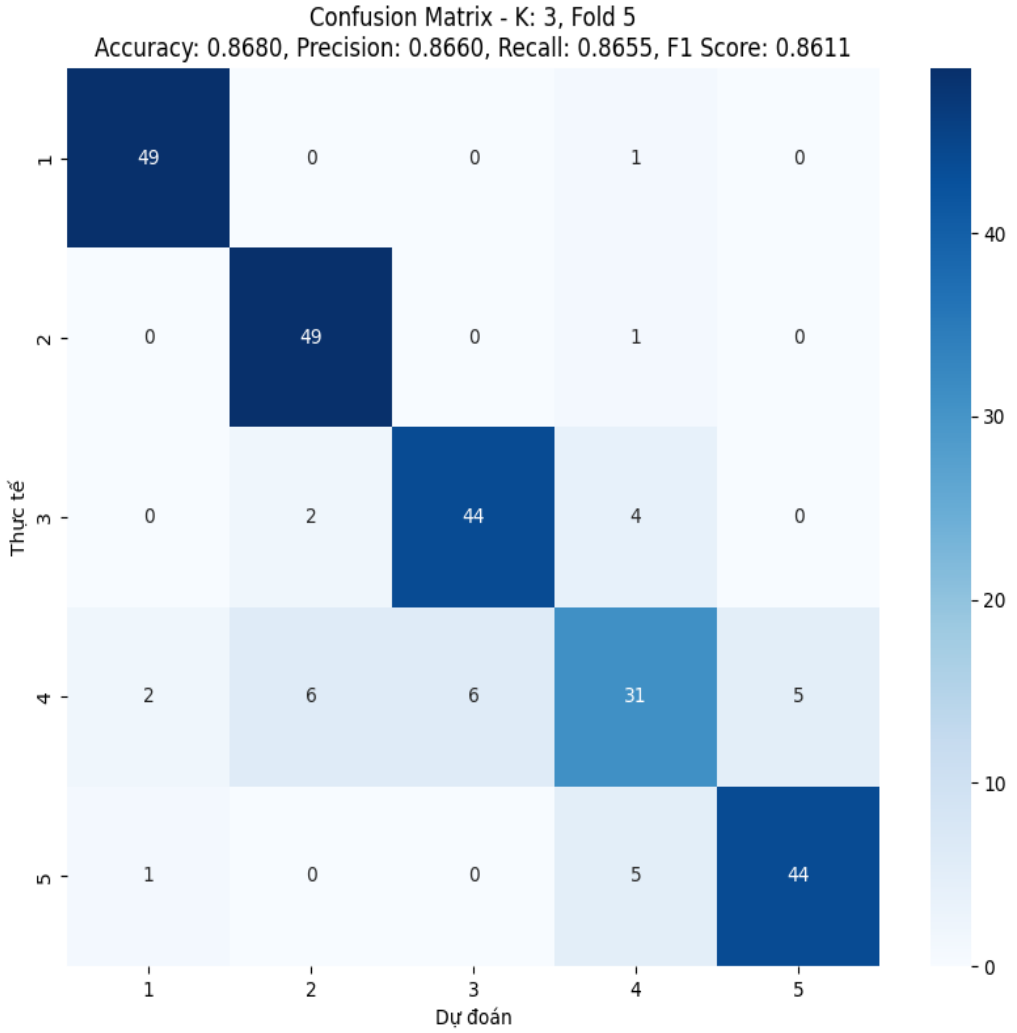
```
k_values = [1, 3, 5, 7]

# Biến lưu kết quả
results = []

# Bước 3 vòng lặp qua các giá trị K
for k in k_values:
    print(f"\nThử nghiệm với K = {k}")

    # Khởi tạo mô hình KNN với K hàng xóm
    classifier = KNeighborsClassifier(n_neighbors=k, metric='minkowski', p=2)

    # Khởi tạo biến lưu trữ kết quả trung bình cho các chỉ số hiệu suất
    accuracy_tb = precision_tb = recall_tb = f1_tb = 0
    cm = np.zeros((num_classes, num_classes))  # Ma trận nhầm lẫn tổng có kích thước cố định

    fold = 0
    for train_index, test_index in kf.split(X):
        fold += 1
        # Chia dữ liệu cho từng fold
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        # Huấn luyện mô hình KNN
        classifier.fit(X_train, y_train)

        # Dự đoán trên tập kiểm thử
        y_pred = classifier.predict(X_test)

        # Hiển thị báo cáo phân loại cho từng fold
        print(f"Results for fold {fold} with K = {k}:")
        print(classification_report(y_test, y_pred, target_names=all_labels))

        # Ma trận nhầm lẫn cho từng fold
        cm_fold = confusion_matrix(y_test, y_pred, labels=all_labels)
        cm += cm_fold  # Cộng dồn ma trận nhầm lẫn, đảm bảo kích thước cố định

        # Tính toán các chỉ số hiệu suất
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='macro')
        recall = recall_score(y_test, y_pred, average='macro')
        f1 = f1_score(y_test, y_pred, average='macro')
```

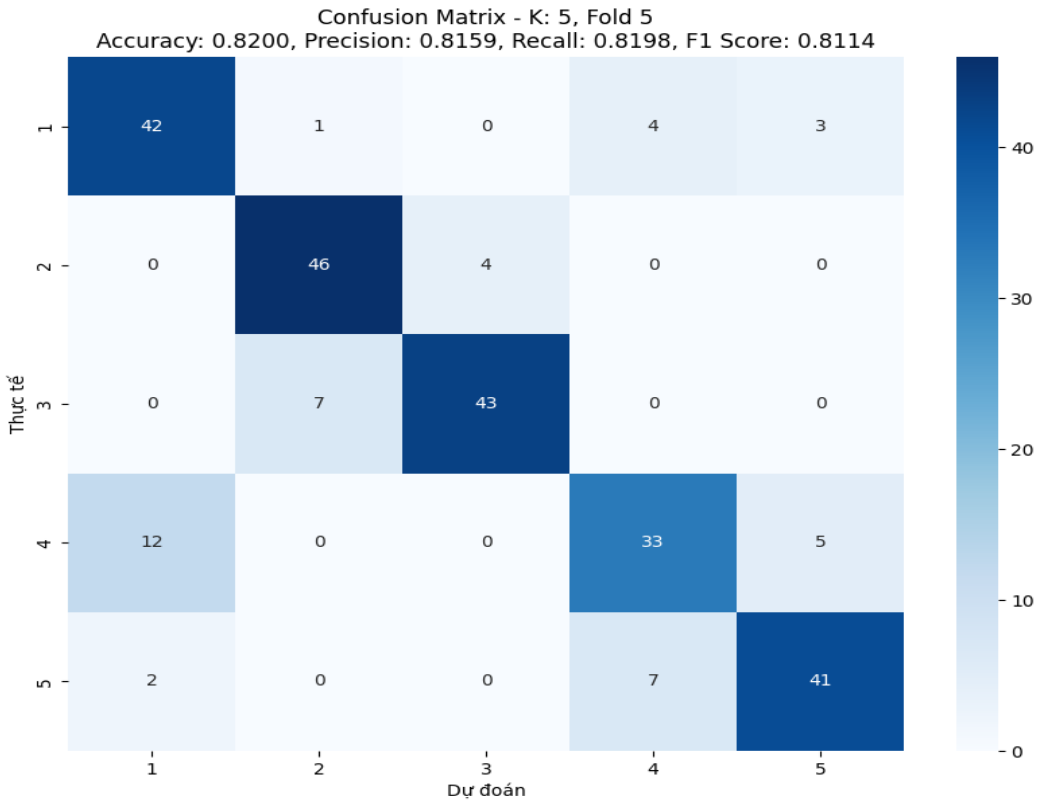
```
        # Hiển thị các chỉ số cho từng fold
        print(f"Accuracy for fold {fold}: {accuracy:.4f}")
        print(f"Precision for fold {fold}: {precision:.4f}")
        print(f"Recall for fold {fold}: {recall:.4f}")
        print(f"F1 score for fold {fold}: {f1:.4f}")
        print("-------------------------------------------------")

        # Cộng dồn các chỉ số để tính trung bình sau tất cả các fold
        accuracy_tb += accuracy
        precision_tb += precision
        recall_tb += recall
        f1_tb += f1

    # Tính và hiển thị các chỉ số hiệu suất trung bình cho tất cả các fold
    accuracy_tb /= fold
    precision_tb /= fold
    recall_tb /= fold
    f1_tb /= fold

    # Lưu kết quả cho giá trị K hiện tại
    results.append({
        'K': k,
        'fold': fold,
        'accuracy': accuracy_tb,
        'precision': precision_tb,
        'recall': recall_tb,
        'f1': f1_tb,
        'confusion_matrix': cm  # Lưu ma trận nhầm lẫn tổng
    })

    print(f"K = {k} - Accuracy average: {accuracy_tb:.4f}")
    print(f"K = {k} - Precision average: {precision_tb:.4f}")
    print(f"K = {k} - Recall average: {recall_tb:.4f}")
    print(f"K = {k} - F1 score average: {f1_tb:.4f}")

    # Vẽ ma trận nhầm lẫn đẹp mắt với Seaborn
    class_labels = [str(label) for label in all_labels]  # Nhãn lớp
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='g', xticklabels=class_labels,
yticklabels=class_labels, cmap='Blues')
    plt.title(f'Confusion Matrix - K: {k}, Fold {fold}\nAccuracy: {accuracy_tb:.4f},
Precision: {precision_tb:.4f}, Recall: {recall_tb:.4f}, F1 Score: {f1_tb:.4f}')
```

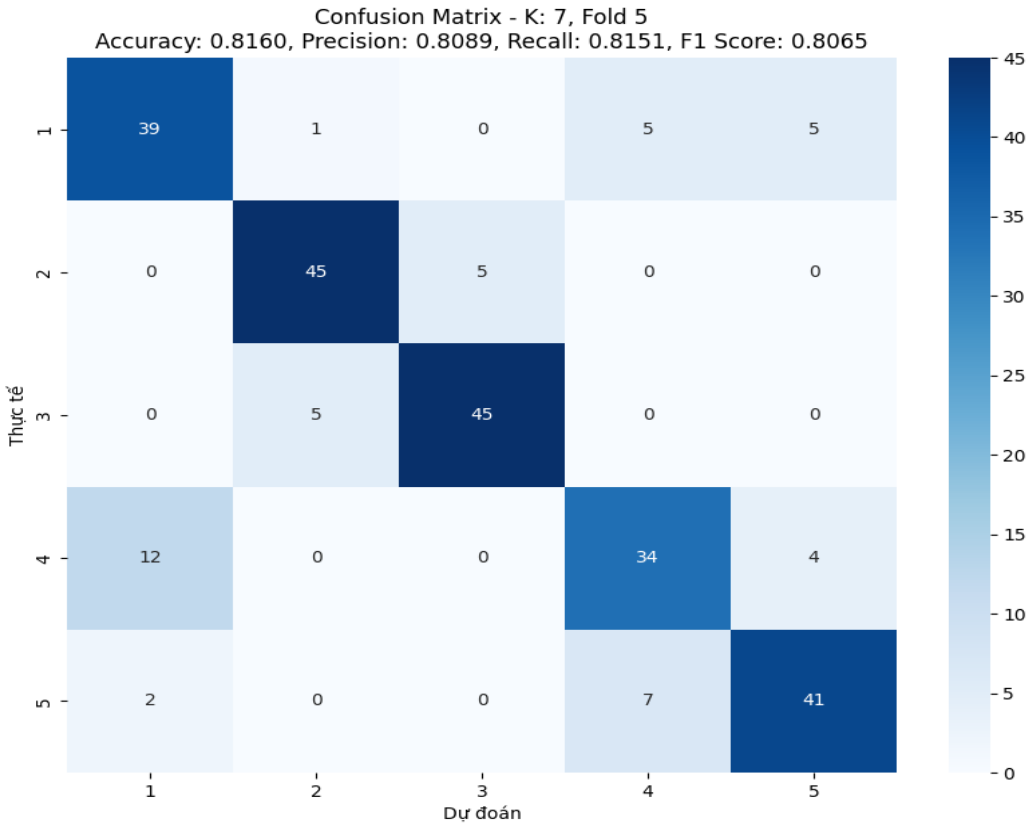
```
    plt.xlabel('Dự đoán')
    plt.ylabel('Thực tế')
    plt.savefig(f'confusion_matrix_k_{k}_fold_{fold}.png')
    plt.close()  # Đóng biểu đồ để tránh hiện thị đè lên nhau

# Hiển thị kết quả tổng quan cho tất cả các giá trị K đã thử nghiệm
print("\nTổng quan kết quả:")
for result in results:
    print(f"K = {result['K']} - Accuracy: {result['accuracy']:.4f}, Precision:
{result['precision']:.4f}, Recall: {result['recall']:.4f}, F1 Score: {result['f1']:.4f}")

# Lưu kết quả vào file CSV
results_df = pd.DataFrame(results)
results_df.to_csv('results_KNN_HU.csv', index=False)
```

Ma trận nhầm lẫn:




Confusion Matrix - K: 3, Fold 5
Accuracy: 0.8680, Precision: 0.8660, Recall: 0.8655, F1 Score: 0.8611
Thực tế
Dự đoán
1
2
3
4
5
49 0 0 1 0
0 49 0 1 0
0 2 44 4 0
2 6 6 31 5
1 0 0 5 44
40
30
20
10
0


Confusion Matrix - K: 5, Fold 5
Accuracy: 0.8200, Precision: 0.8159, Recall: 0.8198, F1 Score: 0.8114
Thực tế
Dự đoán
1 2 3 4 5
42 1 0 4 3
0 46 4 0 0
0 7 43 0 0
12 0 0 33 5
2 0 0 7 41
0 10 20 30 40


Confusion Matrix - K: 7, Fold 5
Accuracy: 0.8160, Precision: 0.8089, Recall: 0.8151, F1 Score: 0.8065
Thực tế
Dự đoán
1 2 3 4 5
39 1 0 5 5
0 45 5 0 0
0 5 45 0 0
12 0 0 34 4
2 0 0 7 41
0 5 10 15 20 25 30 35 40 45

Kết quả được lưu vào file csv tổng quát.

	A	B	C	D	E	F	G	H
1	K	fold	accuracy	precision	recall	f1	confusion_matrix	
2	1	5	0.92	0.921672	0.918724	0.914882	[[49. 0.	
3	3	5	0.868	0.865955	0.865502	0.861118	[[49. 0.	
4	5	5	0.84	0.845946	0.837153	0.828142	[[47. 0.	
5	7	5	0.828	0.830392	0.820755	0.812616	[[47. 0.	

Nhận xét về hệ thống sử dụng KNN với Hu's moments:

Khi số lượng K tăng lên, có một số ảnh hưởng đáng chú ý đến hiệu quả của hệ thống phân loại lá cây dựa trên KNN và đặc trưng Hu's Moments. Dưới đây là nhận xét về sự thay đổi của các chỉ số đánh giá như accuracy, precision, recall, f1-score, và confusion matrix khi tăng K, đánh giá qua phương pháp 5-fold cross-validation:

1. Accuracy

Khi K=1, độ chính xác đạt cao nhất ở mức 0.92, cho thấy mô hình có thể phân loại tốt nhất khi dựa vào láng giềng gần nhất (1 láng giềng).

Khi K tăng dần lên (3, 5, 7), độ chính xác giảm xuống, đạt 0.868 ở K=3, 0.84 ở K=5 và 0.828 ở K = 7.

Sự giảm dần trong độ chính xác cho thấy rằng khi K tăng lên, mô hình trở nên ít chính xác hơn, có thể do các láng giềng xa hơn làm giảm độ phân biệt giữa các lớp.

2. Precision:

Giá trị precision giảm dần từ 0.921 khi K=1 xuống còn 0.830 ở K=7.

Điều này phản ánh rằng mô hình có xu hướng tăng xác suất dương tính giả khi K tăng, tức là mô hình dễ dàng gán sai nhãn cho các lớp khác nhau khi xem xét nhiều láng giềng hơn.

3. Recall:

Giá trị recall cũng giảm nhẹ khi K tăng, từ 0.918 ở K=1 xuống còn 0.821 ở K=7.

Điều này cho thấy mô hình dần bỏ sót nhiều mẫu dương tính thực sự hơn khi số lượng láng giềng tăng, dẫn đến việc ít mẫu thuộc lớp dương tính được dự đoán chính xác.

4. F1 Score:

Giá trị F1 Score giảm dần từ 0.915 ở K=1 xuống còn 0.813 ở K=7.

Sự sụt giảm này cho thấy rằng mô hình trở nên kém cân bằng hơn giữa precision và recall khi K tăng, dẫn đến hiệu suất tổng thể kém đi.

5. Confusion Matrix:

Khi K=1, các lớp được phân biệt khá rõ ràng, chỉ có một số ít nhầm lẫn giữa các lớp (như lớp 3 và lớp 4).

Khi K tăng lên, số lượng nhầm lẫn giữa các lớp tăng lên, đặc biệt là ở lớp 3 và lớp 4, cho thấy mô hình gặp khó khăn trong việc phân loại chính xác các lớp có đặc điểm gần nhau.

Các giá trị trong confusion matrix cho thấy rằng khi số lượng láng giềng tăng, mô hình trở nên "lẫn lộn" giữa các lớp gần nhau, từ đó làm giảm hiệu suất tổng thể.

Kết luận:

Khi K tăng, độ chính xác và hiệu suất tổng thể của hệ thống phân loại lá cây KNN-Hu's Moments giảm dần.

Việc tăng K khiến mô hình phải xem xét nhiều láng giềng hơn, điều này có thể giúp giảm hiện tượng overfitting nhưng lại có nguy cơ giảm khả năng phân biệt giữa các lớp.

Để đạt hiệu quả cao nhất, có thể chọn K=1, vì nó đạt độ chính xác, precision, recall và F1 score cao nhất trong các giá trị K được thử nghiệm.

Bước 4: Thực hiện phân loại lá cây dùng phương pháp ANN, hàm kết nối (net function) là hàm tuyến tính, hàm kích hoạt (activation function) là hàm sigmoid, 1 lớp ẩn, tốc độ học là 0.05. Thay đổi số nơ-ron lớp ẩn để thấy ảnh hưởng của số nơ-ron lớp ẩn đến hiệu quả của hệ thống. Đánh giá hệ thống dùng phương pháp 5-fold cross validation.

ANN-HOG:

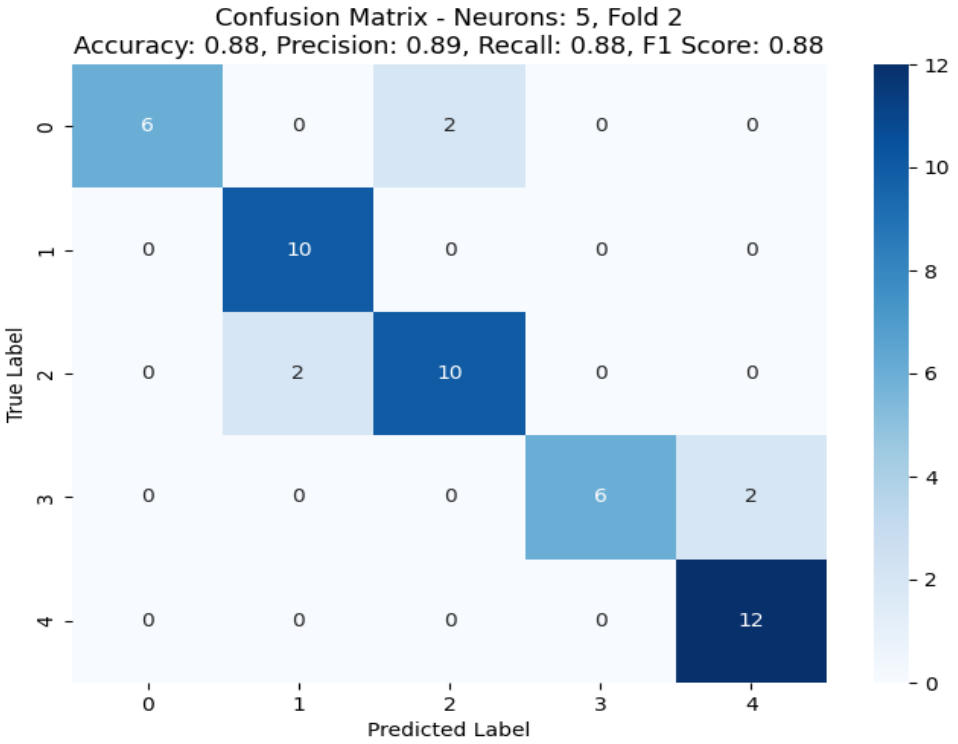
```
# Bước 1: Tải dữ liệu từ file CSV
data = pd.read_csv(r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11Std.csv')

# Giả sử các cột từ 0 đến n-1 là đặc trưng và cột cuối cùng là nhãn
X = data.iloc[:, :-1].values  # Các đặc trưng
y = data.iloc[:, -1].values  # Nhãn

# Kiểm tra giá trị duy nhất trong y
unique_labels = np.unique(y)
print("Các nhãn duy nhất trong dữ liệu:", unique_labels)

# Điều chỉnh nhãn để chúng nằm trong khoảng từ 0 đến num_classes - 1
y = y - unique_labels[0]  # Điều chỉnh nếu nhãn bắt đầu từ 1 hoặc không bắt đầu từ 0

# Chuyển đổi nhãn thành dạng one-hot encoding
num_classes = len(np.unique(y))
y_one_hot = np.eye(num_classes)[y.astype(int)]

# Bước 2: K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Danh sách số neuron để thử nghiệm
neurons_list = [5, 10, 15, 20]

# Lưu kết quả cho tất cả các số neuron
results = []

# Bước 3: Chia dữ liệu một lần duy nhất cho tất cả các số neuron
for fold, (train_idx, test_idx) in enumerate(kfold.split(X)):
    print(f"\nFold {fold + 1}")

    # Chia dữ liệu theo chỉ số train và test
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y_one_hot[train_idx], y_one_hot[test_idx]

    for n_neurons in neurons_list:
        print(f"\nThử nghiệm với số nơ-ron lớp ẩn: {n_neurons}")

        # Bước 4: Xây dựng mô hình ANN
        model = Sequential()
        model.add(Input(shape=(X_train.shape[1],)))  # Sử dụng lớp Input cho đầu vào
        model.add(Dense(n_neurons, activation='sigmoid'))  # Số nơ-ron thay đổi
```

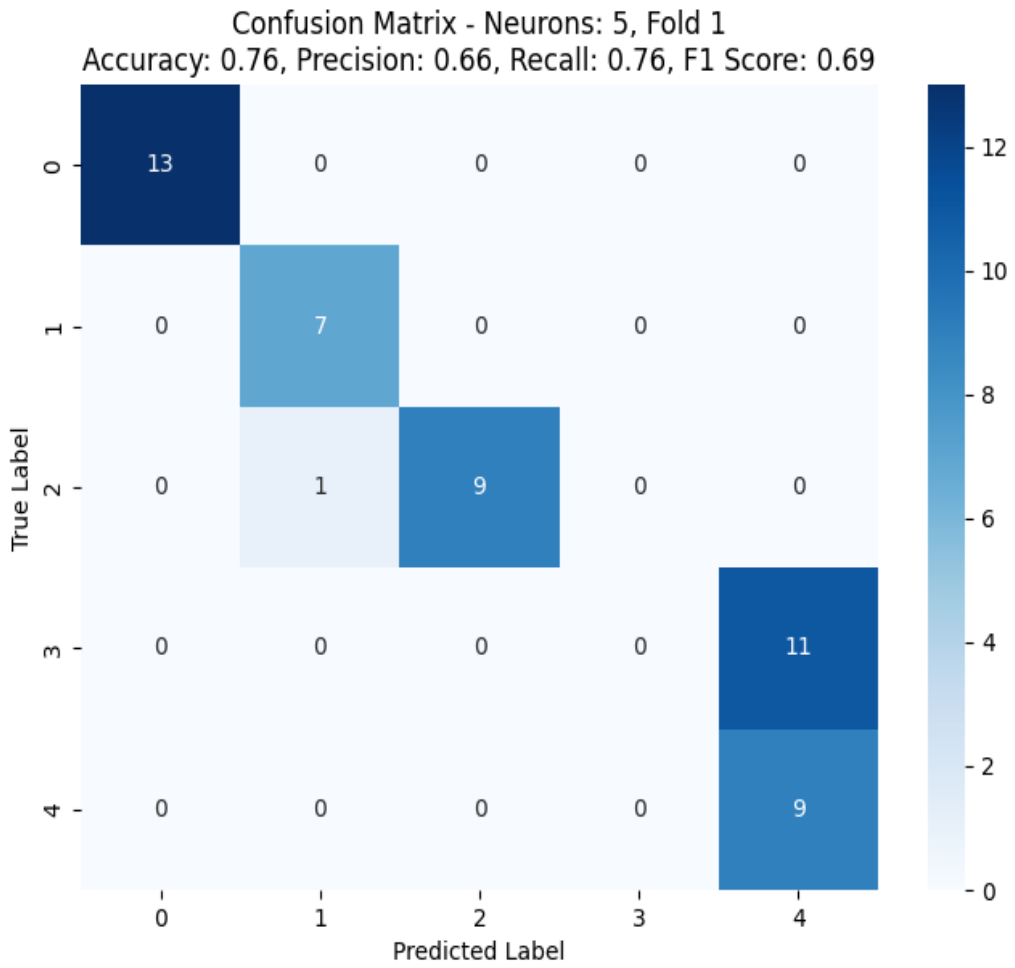
```
        model.add(Dense(num_classes, activation='softmax'))  # Lớp đầu ra

        # Biên dịch mô hình với tốc độ học là 0.05
        model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.05), metrics=['accuracy'])

        # Huấn luyện mô hình
        model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

        # Dự đoán trên tập kiểm tra
        y_pred = model.predict(X_test)
        y_pred_classes = np.argmax(y_pred, axis=1)
        y_test_classes = np.argmax(y_test, axis=1)

        # Tính toán confusion matrix và các chỉ số hiệu suất
        conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)
        accuracy = accuracy_score(y_test_classes, y_pred_classes)
        precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
        recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
        f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')

        # Lưu kết quả cho từng fold
        results.append({
            'neurons': n_neurons,
            'fold': fold + 1,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'confusion_matrix': conf_matrix
        })

        # Hiển thị thông số đánh giá
        print(f'Fold {fold + 1} - Số nơ-ron: {n_neurons} - Accuracy: {accuracy:.2f},
Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}') #2f là làm tròn lên 2
số.
        print('----------------------------------------')

        # Vẽ confusion matrix cho từng fold và lưu vào file
        plt.figure(figsize=(8, 6))
        sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues',xticklabels=np.unique(y_test_classes),
yticklabels=np.unique(y_test_classes))
```

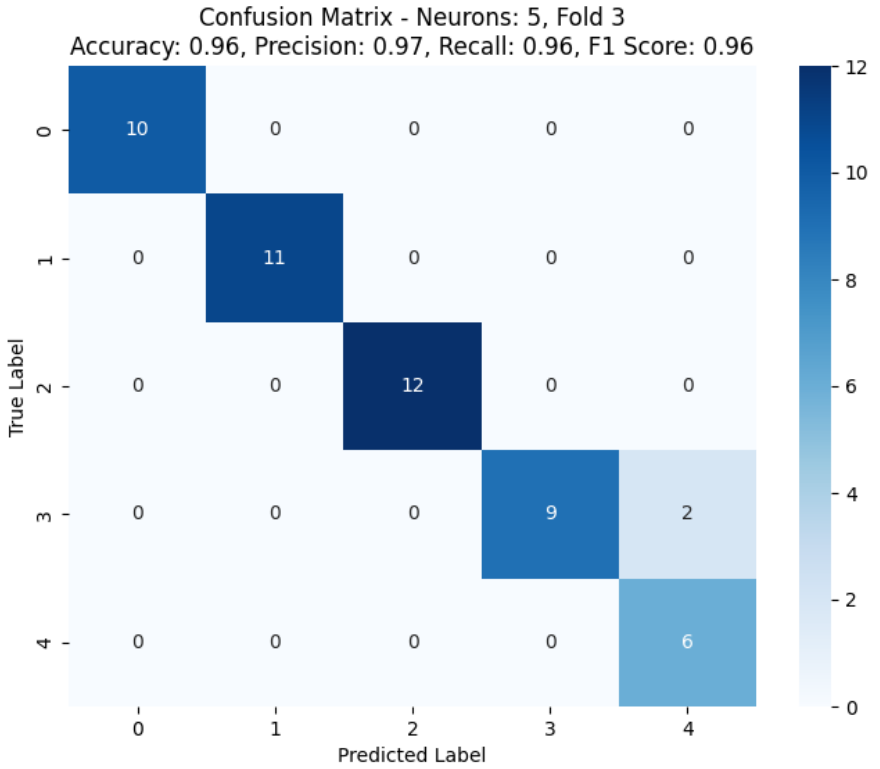
```
        plt.title(f'Confusion Matrix - Neurons: {n_neurons}, Fold {fold + 1}\nAccuracy:
{accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')

        # Lưu hình vào file (đặt tên theo số nơ-ron và số fold)
        plt.savefig(f'confusion_matrix_neurons_{n_neurons}_fold_{fold + 1}.png')
        plt.close()  # Đóng hình để giải phóng bộ nhớ

# Lưu kết quả vào file CSV
results_df = pd.DataFrame(results)
results_df.to_csv('results_ANN_HOG.csv', index=False)
```

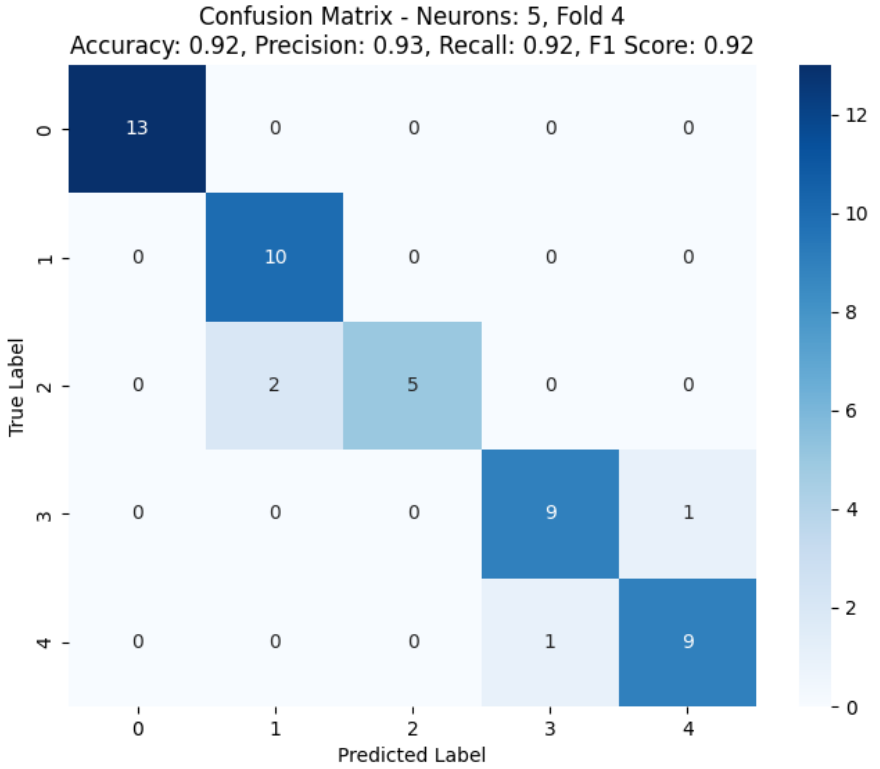
Kết quả:

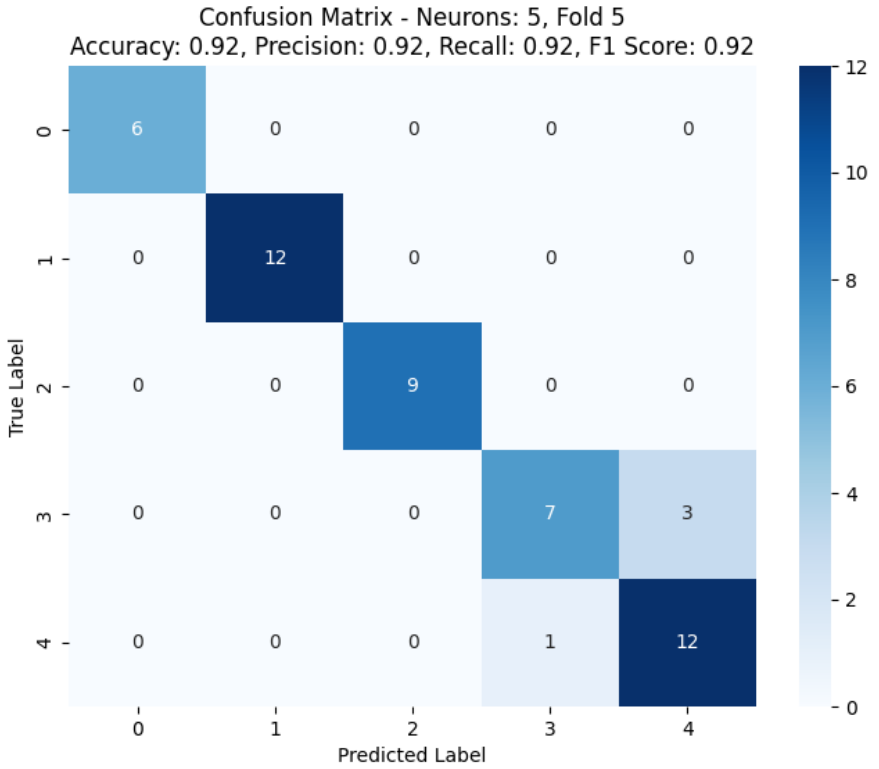
Ma trận nhầm lẫn:

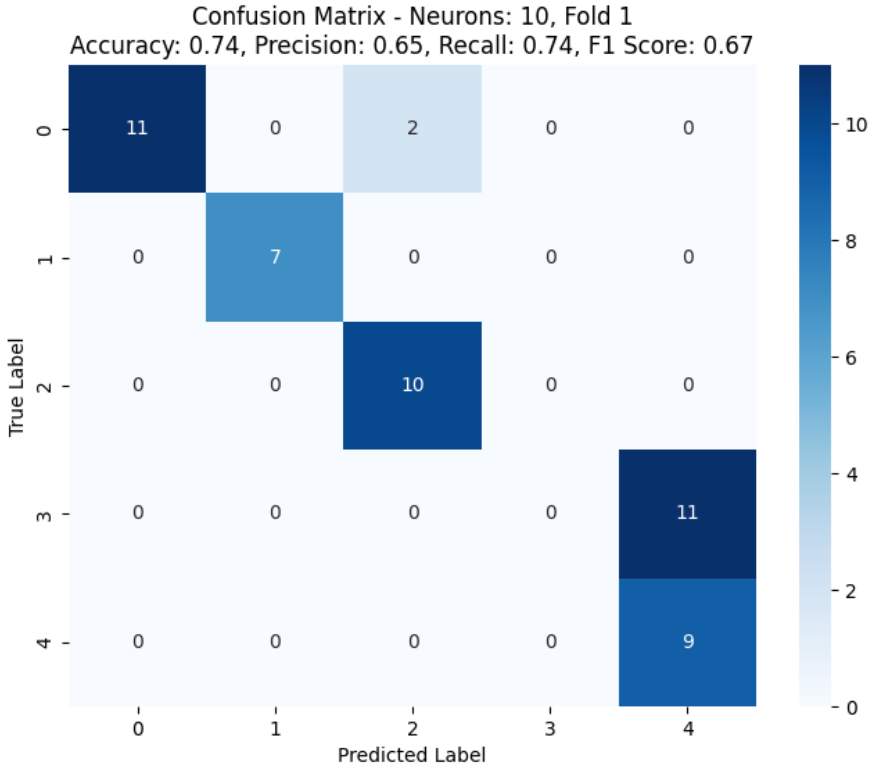

Confusion Matrix - Neurons: 5, Fold 2
Accuracy: 0.88, Precision: 0.89, Recall: 0.88, F1 Score: 0.88
True Label
Predicted Label
6 0 2 0 0
0 10 0 0 0
0 2 10 0 0
0 0 0 6 2
0 0 0 0 12
0 1 2 3 4
12 10 8 6 4 2 0

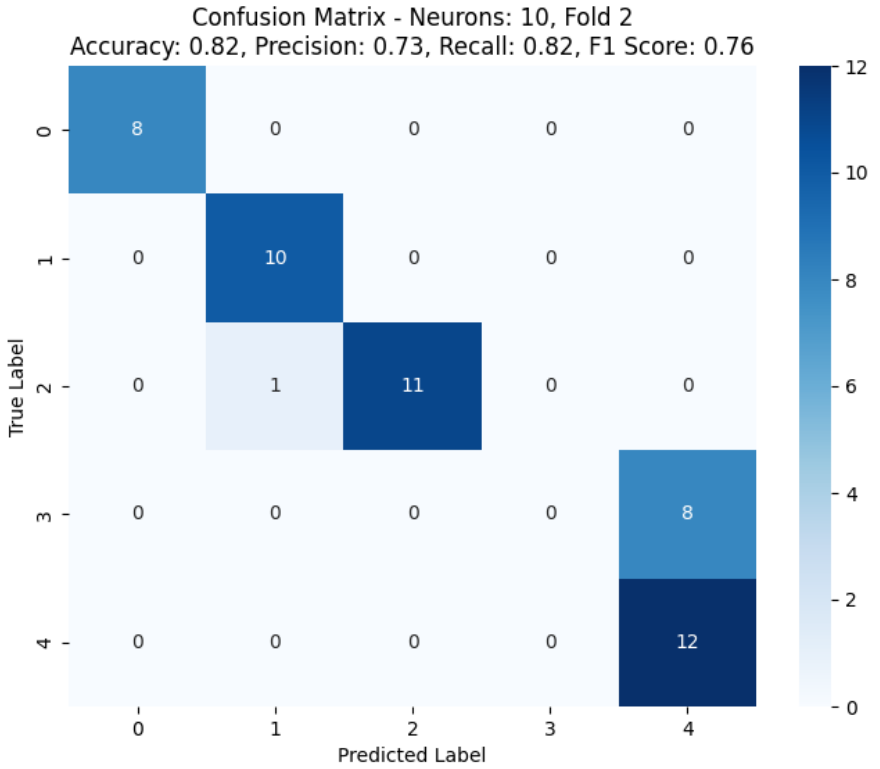

Confusion Matrix - Neurons: 5, Fold 1
Accuracy: 0.76, Precision: 0.66, Recall: 0.76, F1 Score: 0.69
True Label
Predicted Label
13 0 0 0 0
0 7 0 0 0
0 1 9 0 0
0 0 0 0 11
0 0 0 0 9
0 1 2 3 4
12 10 8 6 4 2 0

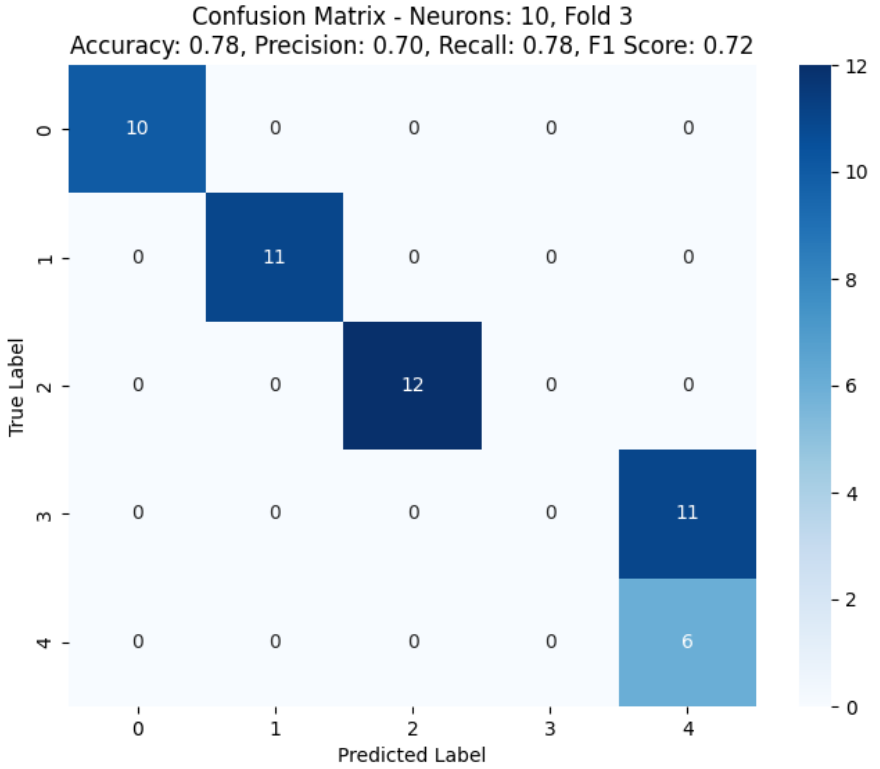

Confusion Matrix - Neurons: 5, Fold 3
Accuracy: 0.96, Precision: 0.97, Recall: 0.96, F1 Score: 0.96
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 9 2
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12

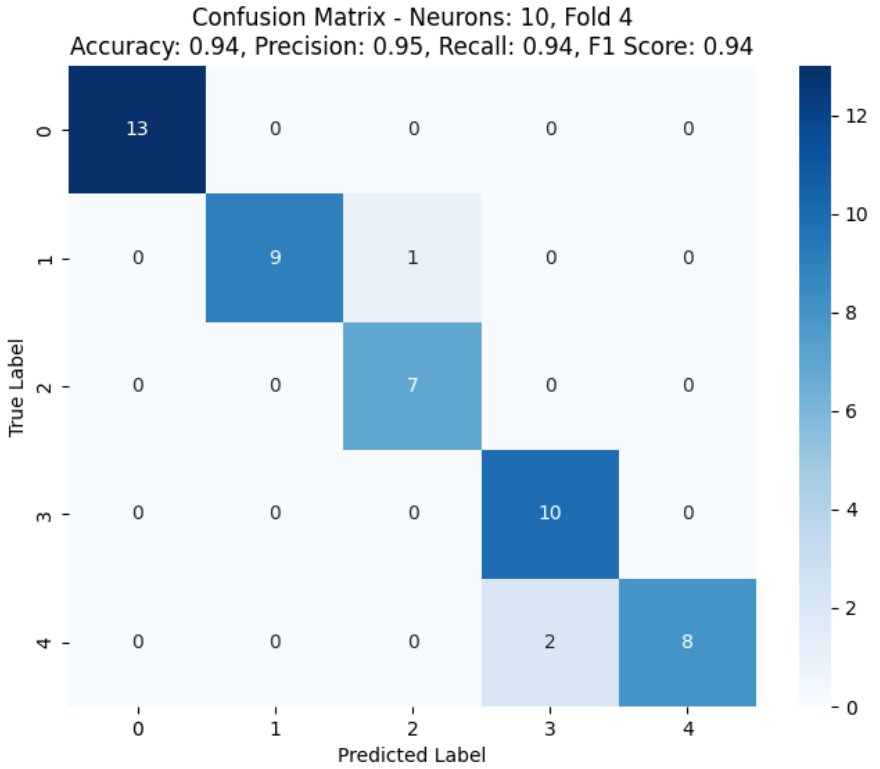

Confusion Matrix - Neurons: 5, Fold 4
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
13 0 0 0 0
0 10 0 0 0
0 2 5 0 0
0 0 0 9 1
0 0 0 1 9
0 1 2 3 4
0 2 4 6 8 10 12

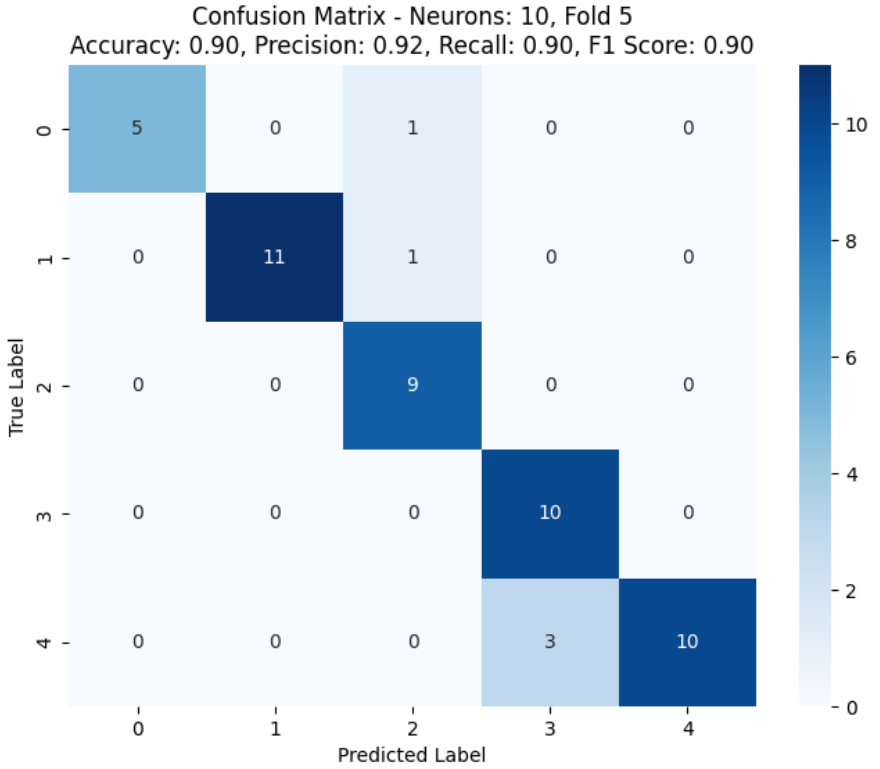

Confusion Matrix - Neurons: 5, Fold 5
Accuracy: 0.92, Precision: 0.92, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 0 0 7 3
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12

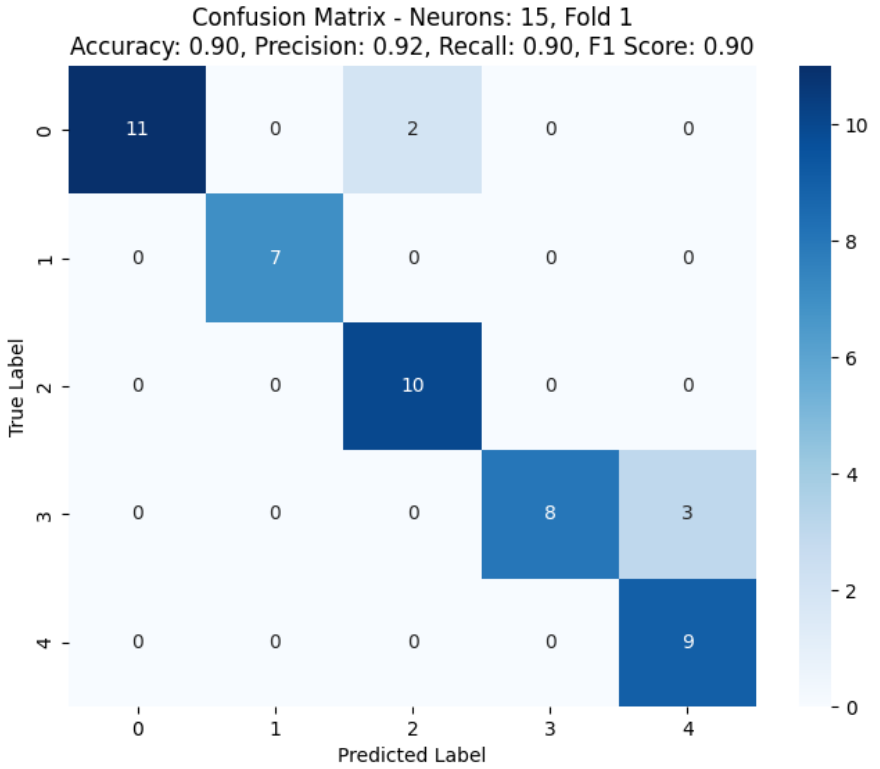

Confusion Matrix - Neurons: 10, Fold 1
Accuracy: 0.74, Precision: 0.65, Recall: 0.74, F1 Score: 0.67
True Label
Predicted Label
11 0 2 0 0
0 7 0 0 0
0 0 10 0 0
0 0 0 0 11
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10

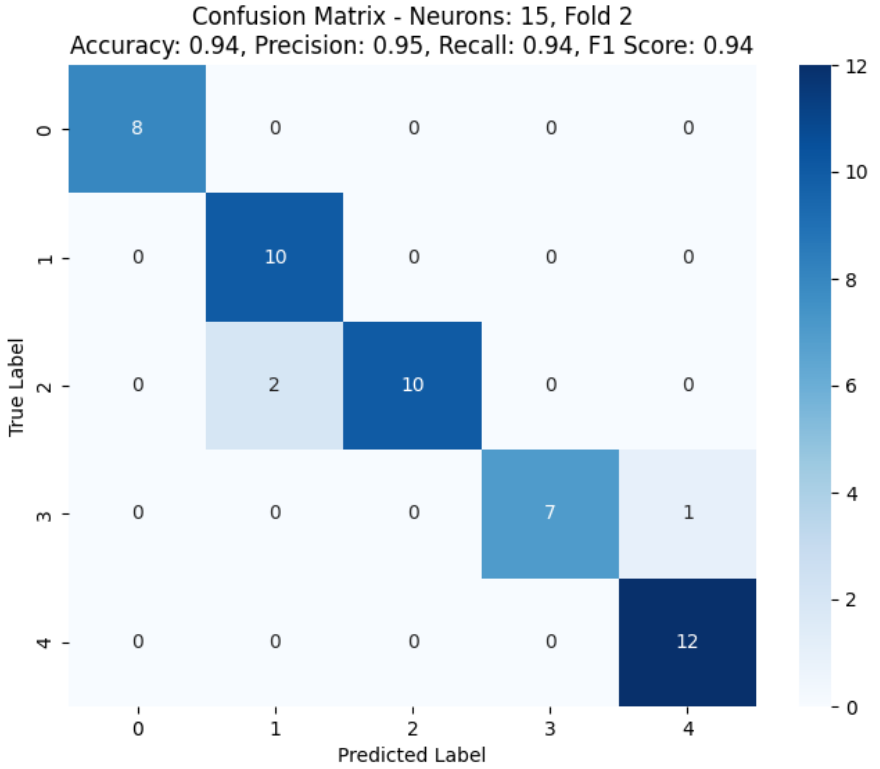

Confusion Matrix - Neurons: 10, Fold 2
Accuracy: 0.82, Precision: 0.73, Recall: 0.82, F1 Score: 0.76
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 1 11 0 0
0 0 0 0 8
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

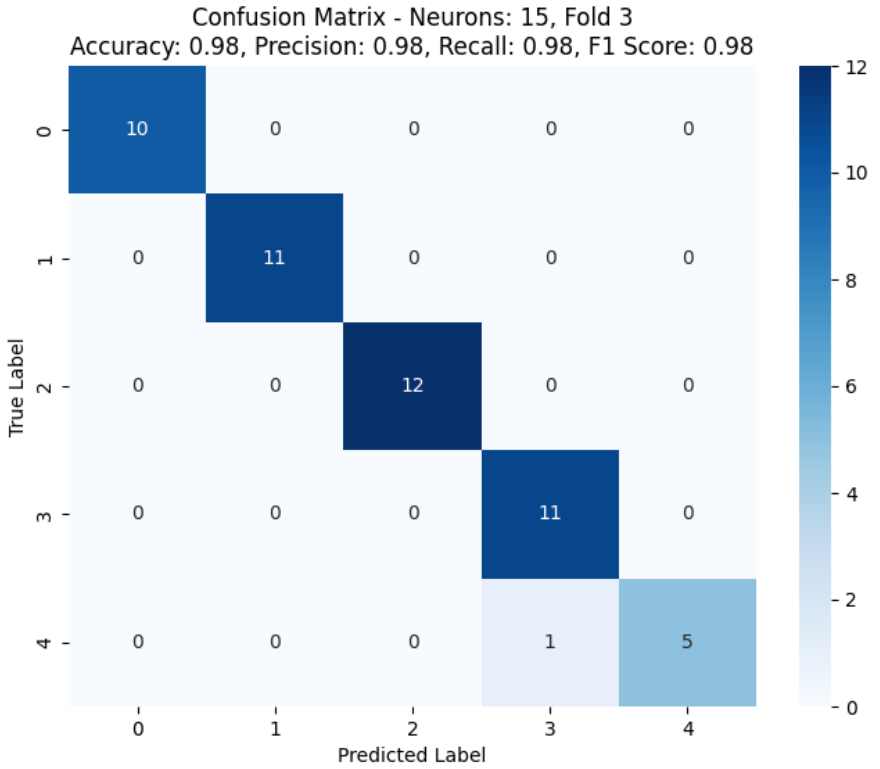

Confusion Matrix - Neurons: 10, Fold 3
Accuracy: 0.78, Precision: 0.70, Recall: 0.78, F1 Score: 0.72
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 0 11
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12

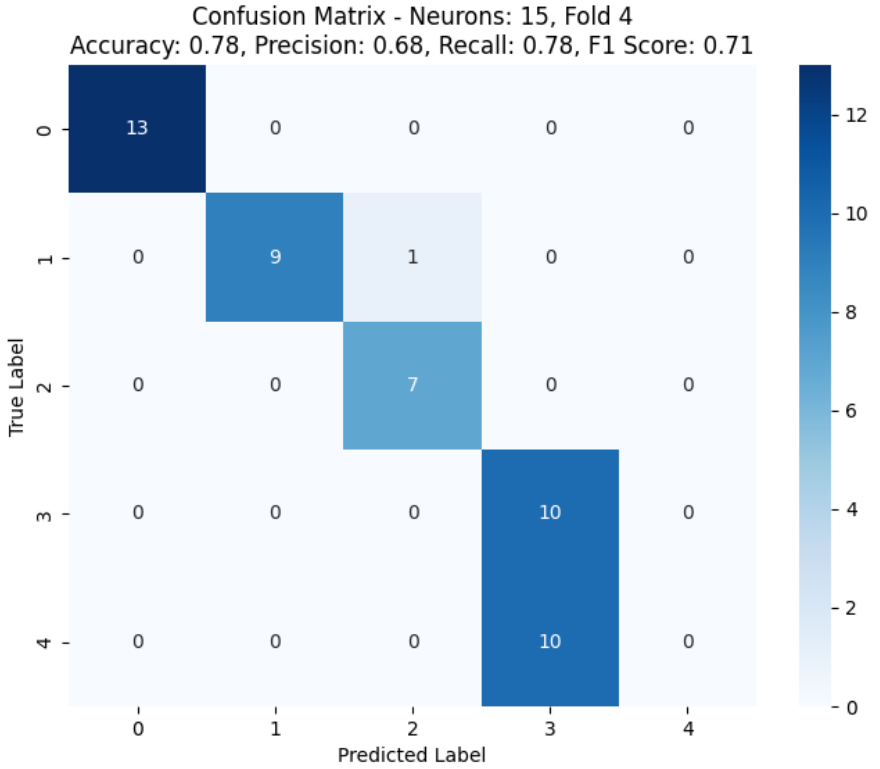

Confusion Matrix - Neurons: 10, Fold 4
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
0 0 0 10 0
0 0 0 2 8
0 1 2 3 4
0 2 4 6 8 10 12

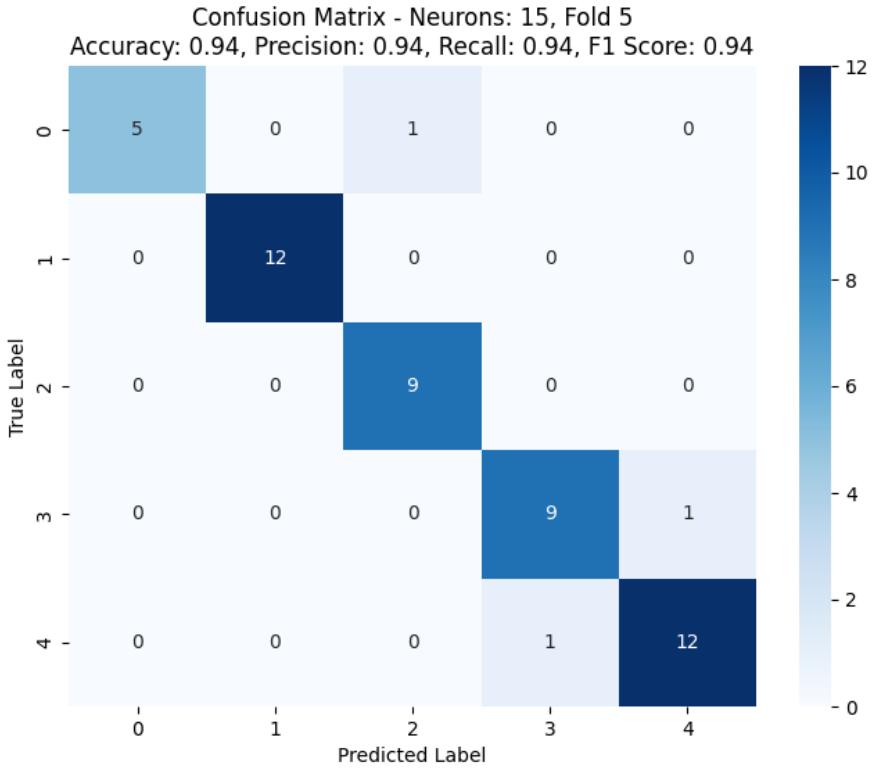

Confusion Matrix - Neurons: 10, Fold 5
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
5 0 1 0 0
0 11 1 0 0
0 0 9 0 0
0 0 0 10 0
0 0 0 3 10
0 1 2 3 4
0 2 4 6 8 10

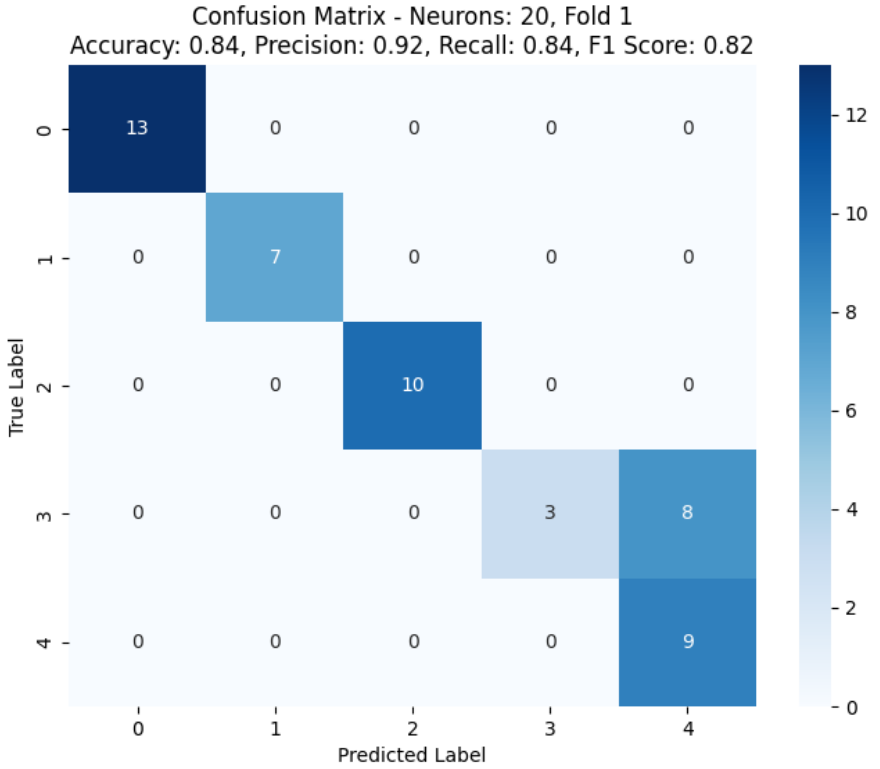

Confusion Matrix - Neurons: 15, Fold 1
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
11 0 2 0 0
0 7 0 0 0
0 0 10 0 0
0 0 0 8 3
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10

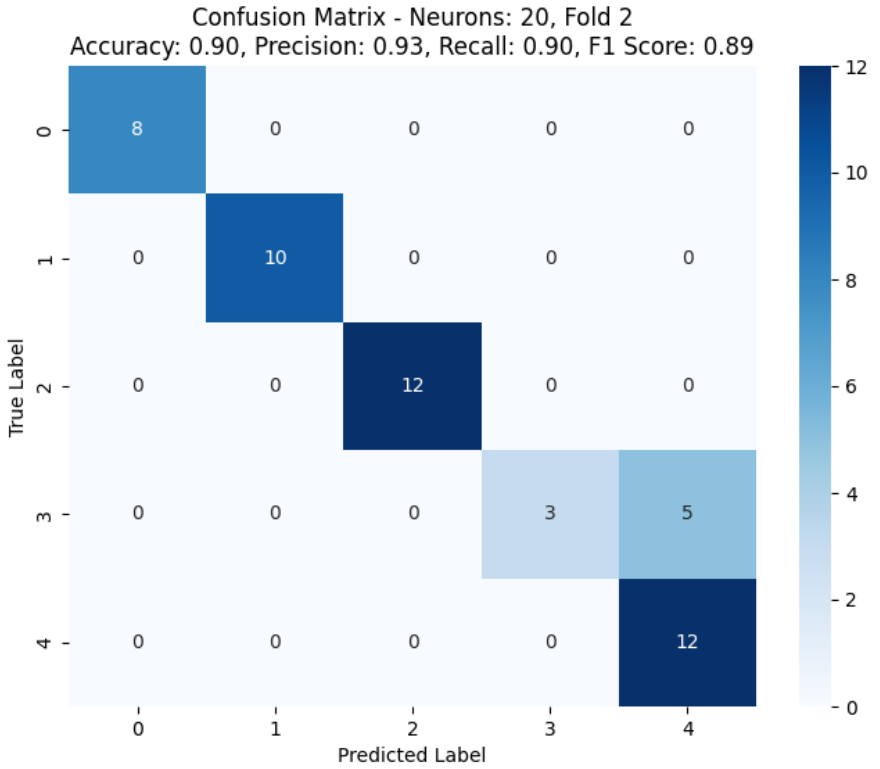

Confusion Matrix - Neurons: 15, Fold 2
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 2 10 0 0
0 0 0 7 1
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

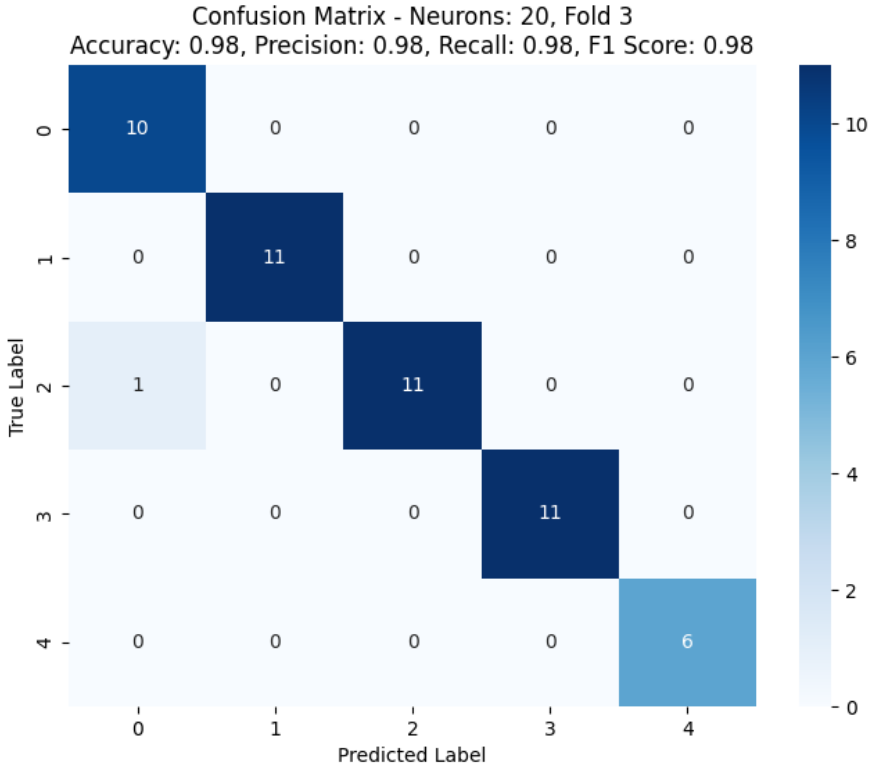

Confusion Matrix - Neurons: 15, Fold 3
Accuracy: 0.98, Precision: 0.98, Recall: 0.98, F1 Score: 0.98
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 11 0
0 0 0 1 5
0 1 2 3 4
0 2 4 6 8 10 12

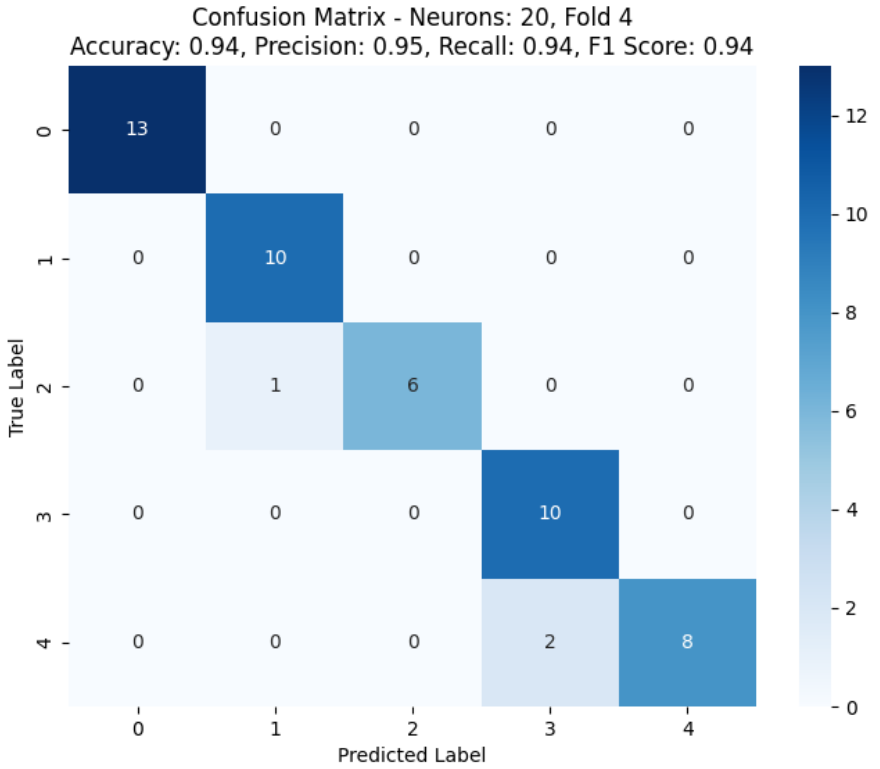

Confusion Matrix - Neurons: 15, Fold 4
Accuracy: 0.78, Precision: 0.68, Recall: 0.78, F1 Score: 0.71
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
0 0 0 10 0
0 0 0 10 0
0 1 2 3 4
0 2 4 6 8 10 12

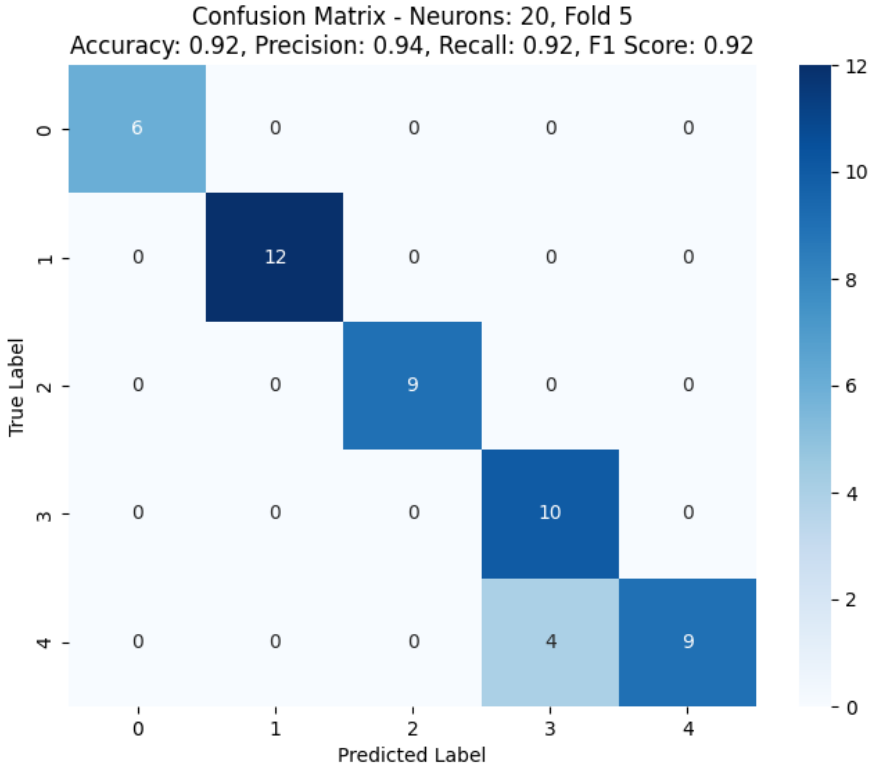

Confusion Matrix - Neurons: 15, Fold 5
Accuracy: 0.94, Precision: 0.94, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
5 0 1 0 0
0 12 0 0 0
0 0 9 0 0
0 0 0 9 1
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 1
Accuracy: 0.84, Precision: 0.92, Recall: 0.84, F1 Score: 0.82
True Label
Predicted Label
13 0 0 0 0
0 7 0 0 0
0 0 10 0 0
0 0 0 3 8
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 2
Accuracy: 0.90, Precision: 0.93, Recall: 0.90, F1 Score: 0.89
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 0 12 0 0
0 0 0 3 5
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 3
Accuracy: 0.98, Precision: 0.98, Recall: 0.98, F1 Score: 0.98
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
1 0 11 0 0
0 0 0 11 0
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10


Confusion Matrix - Neurons: 20, Fold 4
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
13 0 0 0 0
0 10 0 0 0
0 1 6 0 0
0 0 0 10 0
0 0 0 2 8
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 5
Accuracy: 0.92, Precision: 0.94, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 0 0 10 0
0 0 0 4 9
0 1 2 3 4
0 2 4 6 8 10 12

Bảng csv kết quả tổng quát

	A	B	C	D	E	F	G	H
1	neurons	fold	accuracy	precision	recall	f1	confusion_matrix	
2	5	1	0.8	0.905263	0.8	0.752381	[[13 0 0	
3	10	1	0.84	0.882381	0.84	0.836685	[[12 0 1	
4	15	1	0.92	0.9375	0.92	0.919689	[[13 0 0	
5	20	1	0.94	0.940833	0.94	0.93954	[[13 0 0	
6	5	2	0.64	0.434909	0.64	0.509412	[[8 0 0	
7	10	2	0.9	0.915152	0.9	0.900752	[[8 0 0	
8	15	2	0.82	0.725818	0.82	0.760041	[[8 0 0	
9	20	2	0.98	0.982222	0.98	0.980153	[[8 0 0	
10	5	3	0.76	0.684171	0.76	0.70265	[[10 0 0	
11	10	3	0.94	0.945152	0.94	0.938618	[[10 0 0	
12	15	3	0.9	0.905487	0.9	0.898426	[[10 0 0	
13	20	3	1	1	1	1	[[10 0 0	
14	5	4	0.98	0.981818	0.98	0.97995	[[13 0 0	
15	10	4	0.98	0.9825	0.98	0.98014	[[13 0 0	
16	15	4	0.88	0.915152	0.88	0.87304	[[13 0 0	
17	20	4	0.92	0.923232	0.92	0.919557	[[13 0 0	
18	5	5	0.62	0.546957	0.62	0.523212	[[6 0 0	
19	10	5	0.8	0.686957	0.8	0.727778	[[6 0 0	
20	15	5	0.96	0.965333	0.96	0.959206	[[6 0 0	
21	20	5	1	1	1	1	[[6 0 0	

Kết quả trung bình của các chỉ số cho từng số lượng neuron là:

- Neuron 5:
  - Accuracy: 0.760
  - Precision: 0.7106
  - Recall: 0.760
  - F1: 0.6935
- Neuron 10:
  - Accuracy: 0.892
  - Precision: 0.8824
  - Recall: 0.892
  - F1: 0.8768
- Neuron 15:
  - Accuracy: 0.896

    - Precision: 0.8899
    - Recall: 0.896
    - F1: 0.8821
- Neuron 20:
    - Accuracy: 0.968
    - Precision: 0.9693
    - Recall: 0.968
    - F1: 0.9679

Dựa trên các giá trị trung bình của các chỉ số (accuracy, precision, recall, f1) khi thay đổi số lượng neuron từ 5 đến 20, chúng ta có thể nhận xét về ảnh hưởng của số lượng neuron đến hiệu quả của hệ thống như sau:

1. Tăng số lượng neuron giúp cải thiện hiệu quả:
    - Khi số lượng neuron tăng từ 5 đến 20, các chỉ số hiệu quả (accuracy, precision, recall, và f1) đều tăng lên. Điều này cho thấy rằng việc tăng số lượng neuron giúp hệ thống có khả năng phân loại tốt hơn, thể hiện qua sự tăng đều của các chỉ số.
2. Cải thiện rõ rệt khi số neuron tăng từ 5 lên 10 và 15:
    - Ở mức 5 neuron, các chỉ số đều thấp nhất, cho thấy mô hình còn hạn chế trong việc phân loại chính xác. Khi số neuron tăng lên 10 và 15, các chỉ số tăng đáng kể, đặc biệt là accuracy và precision. Điều này có thể do mô hình với số neuron ít bị thiếu khả năng học các đặc trưng phức tạp của dữ liệu, nhưng khi tăng số neuron lên, mô hình có thể học tốt hơn.
3. Đạt hiệu quả cao nhất với 20 neuron:
    - Với 20 neuron, các chỉ số đều đạt mức cao nhất (accuracy = 0.968, precision = 0.9693, recall = 0.968, f1 = 0.9679). Điều này cho thấy ở mức 20 neuron, mô hình có thể đã đạt đến khả năng phân loại tối ưu trên tập dữ liệu này.
4. Sự cải thiện có thể chững lại ở mức neuron cao hơn:
    - Mặc dù hiệu quả tăng lên khi tăng số neuron, nhưng sau một ngưỡng nhất định (như từ 15 lên 20), sự cải thiện không còn quá lớn. Điều này có thể là dấu hiệu rằng việc tăng thêm neuron có thể không mang lại nhiều cải thiện đáng kể và có thể dẫn đến overfitting khi mô hình trở nên quá phức tạp cho dữ liệu.

Kết luận: Việc tăng số lượng neuron giúp cải thiện hiệu quả hệ thống phân loại, đặc biệt khi tăng từ mức thấp (5 neuron) đến mức trung bình (10–15 neuron). Tuy nhiên, sau mức 20 neuron, hiệu quả có thể chững lại và cần kiểm tra để tránh overfitting.

.

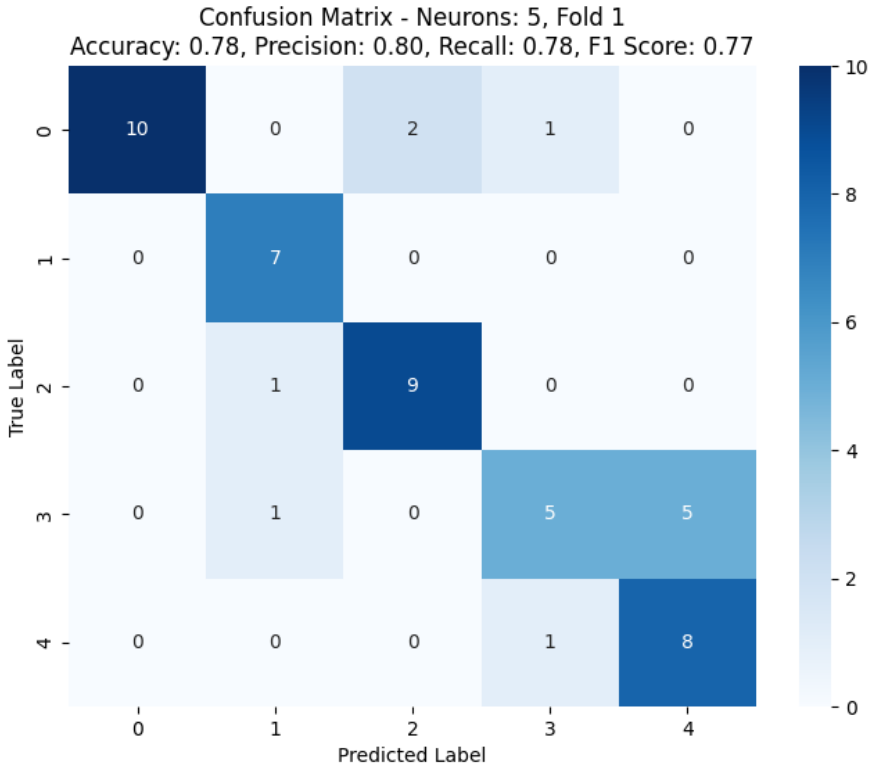
ANN-HU:

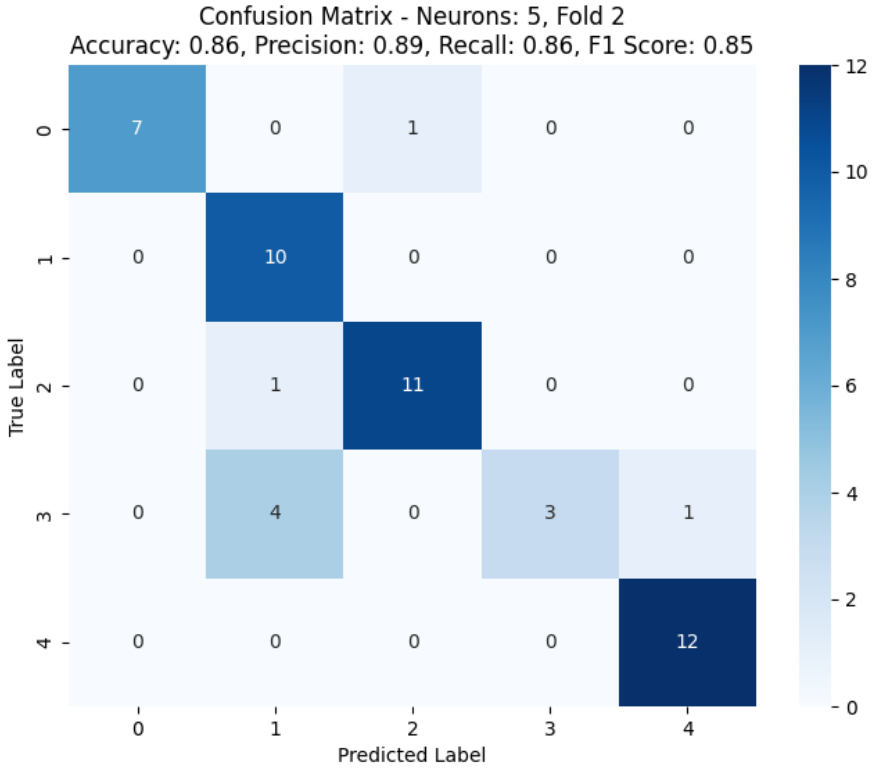
```
# Bước 1: Tải dữ liệu từ file CSV
data = pd.read_csv(r'D:/DATA/Hu/hutest_hsv/HUnhom11Std.csv')

# Giả sử các cột từ 0 đến n-1 là đặc trưng và cột cuối cùng là nhãn
X = data.iloc[:, :-1].values  # Các đặc trưng
y = data.iloc[:, -1].values  # Nhãn

# Kiểm tra giá trị duy nhất trong y
unique_labels = np.unique(y)
print("Các nhãn duy nhất trong dữ liệu:", unique_labels)

# Điều chỉnh nhãn để chúng nằm trong khoảng từ 0 đến num_classes - 1
y = y - unique_labels[0]  # Điều chỉnh nếu nhãn bắt đầu từ 1 hoặc không bắt đầu từ 0

# Chuyển đổi nhãn thành dạng one-hot encoding
num_classes = len(np.unique(y))
y_one_hot = np.eye(num_classes)[y.astype(int)]

# Bước 2: K-fold Cross Validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)

# Danh sách số neuron để thử nghiệm
neurons_list = [5, 10, 15, 20]

# Lưu kết quả cho tất cả các số neuron
results = []

# Bước 3: Chia dữ liệu một lần duy nhất cho tất cả các số neuron
for fold, (train_idx, test_idx) in enumerate(kfold.split(X)):
    print(f"\nFold {fold + 1}")

    # Chia dữ liệu theo chỉ số train và test
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y_one_hot[train_idx], y_one_hot[test_idx]

    for n_neurons in neurons_list:
        print(f"\nThử nghiệm với số nơ-ron lớp ẩn: {n_neurons}")

        # Bước 4: Xây dựng mô hình ANN
        model = Sequential()
        model.add(Input(shape=(X_train.shape[1],)))  # Sử dụng lớp Input cho đầu vào
        model.add(Dense(n_neurons, activation='sigmoid'))  # Số nơ-ron thay đổi
```

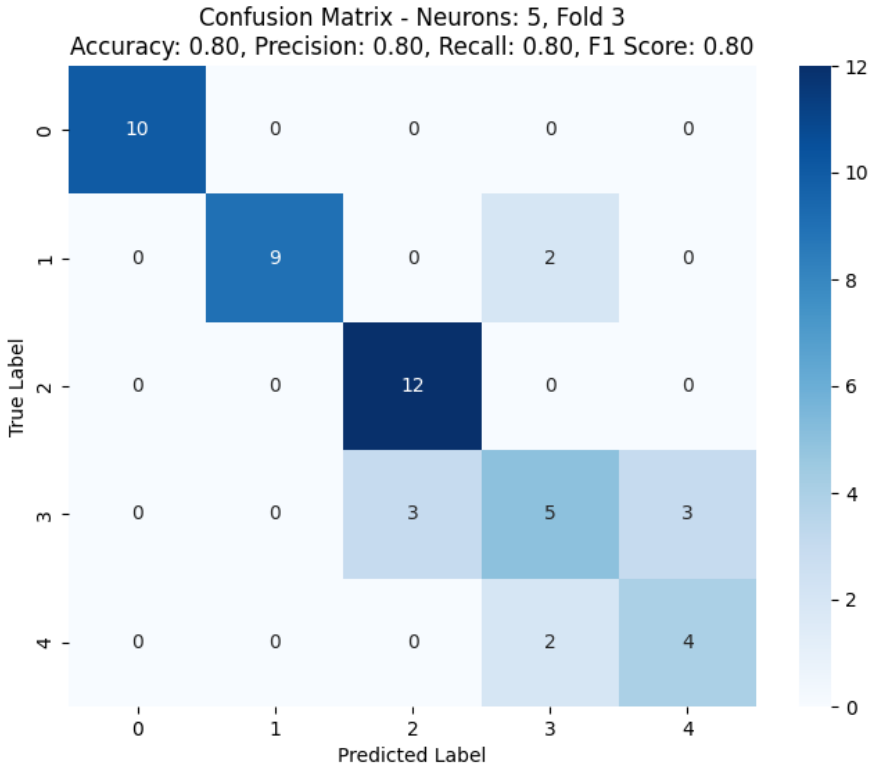
```
        model.add(Dense(num_classes, activation='softmax'))  # Lớp đầu ra

        # Biên dịch mô hình với tốc độ học là 0.05
        model.compile(loss='categorical_crossentropy',
optimizer=Adam(learning_rate=0.05), metrics=['accuracy'])

        # Huấn luyện mô hình
        model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

        # Dự đoán trên tập kiểm tra
        y_pred = model.predict(X_test)
        y_pred_classes = np.argmax(y_pred, axis=1)
        y_test_classes = np.argmax(y_test, axis=1)

        # Tính toán confusion matrix và các chỉ số hiệu suất
        conf_matrix = confusion_matrix(y_test_classes, y_pred_classes)
        accuracy = accuracy_score(y_test_classes, y_pred_classes)
        precision = precision_score(y_test_classes, y_pred_classes, average='weighted')
        recall = recall_score(y_test_classes, y_pred_classes, average='weighted')
        f1 = f1_score(y_test_classes, y_pred_classes, average='weighted')

        # Lưu kết quả cho từng fold
        results.append({
            'neurons': n_neurons,
            'fold': fold + 1,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1': f1,
            'confusion_matrix': conf_matrix
        })

        # Hiển thị thông số đánh giá
        print(f'Fold {fold + 1} - Số nơ-ron: {n_neurons} - Accuracy: {accuracy:.2f},
Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
        print('----------------------------------------')

        # Vẽ confusion matrix cho từng fold và lưu vào file
        plt.figure(figsize=(8, 6))
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                    xticklabels=np.unique(y_test_classes),
yticklabels=np.unique(y_test_classes))
```

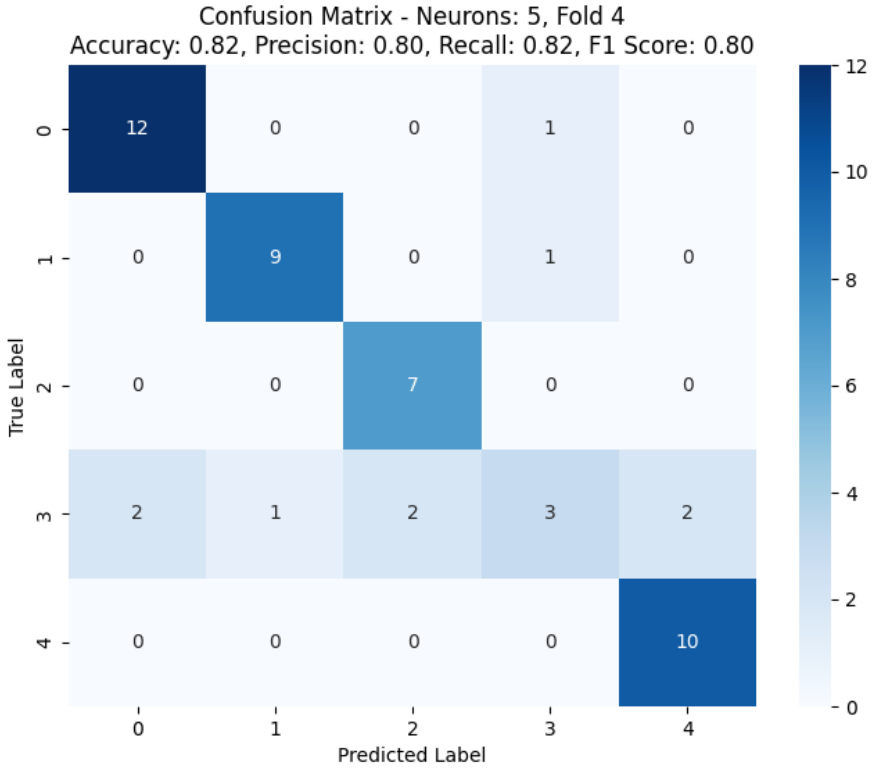
```
        plt.title(f'Confusion Matrix - Neurons: {n_neurons}, Fold {fold + 1}\nAccuracy:
{accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')

        # Lưu hình vào file (đặt tên theo số nơ-ron và số fold)
        plt.savefig(f'confusion_matrix_neurons_{n_neurons}_fold_{fold + 1}.png')
        plt.close()  # Đóng hình để giải phóng bộ nhớ

# Lưu kết quả vào file CSV
results_df = pd.DataFrame(results)
results_df.to_csv('results_ANN_HU.csv', index=False)
```

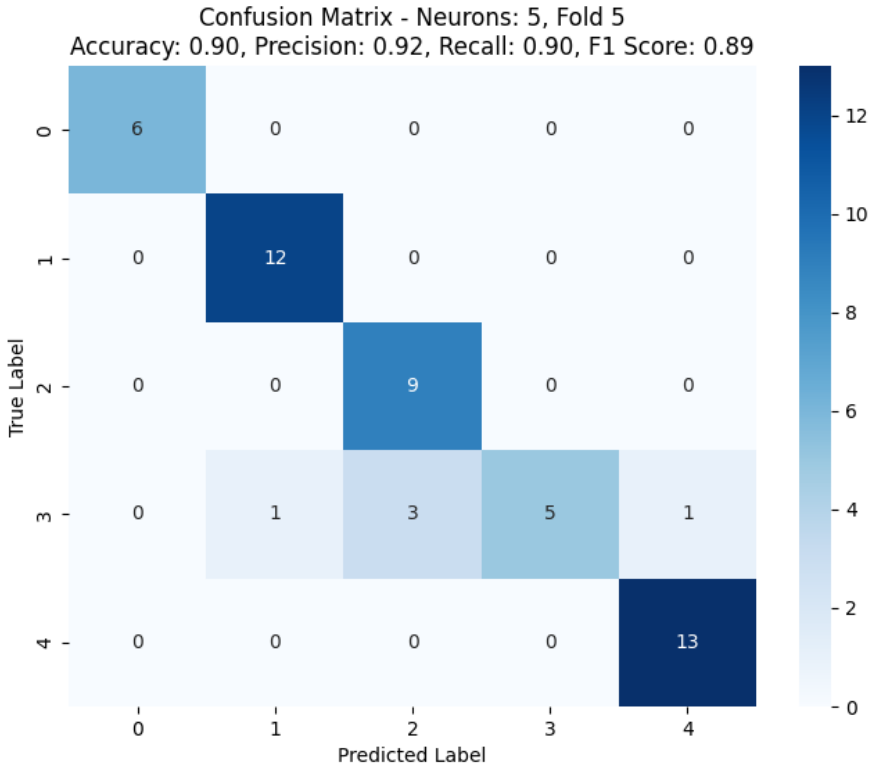
Kết quả:
Ma trận nhầm lẫn

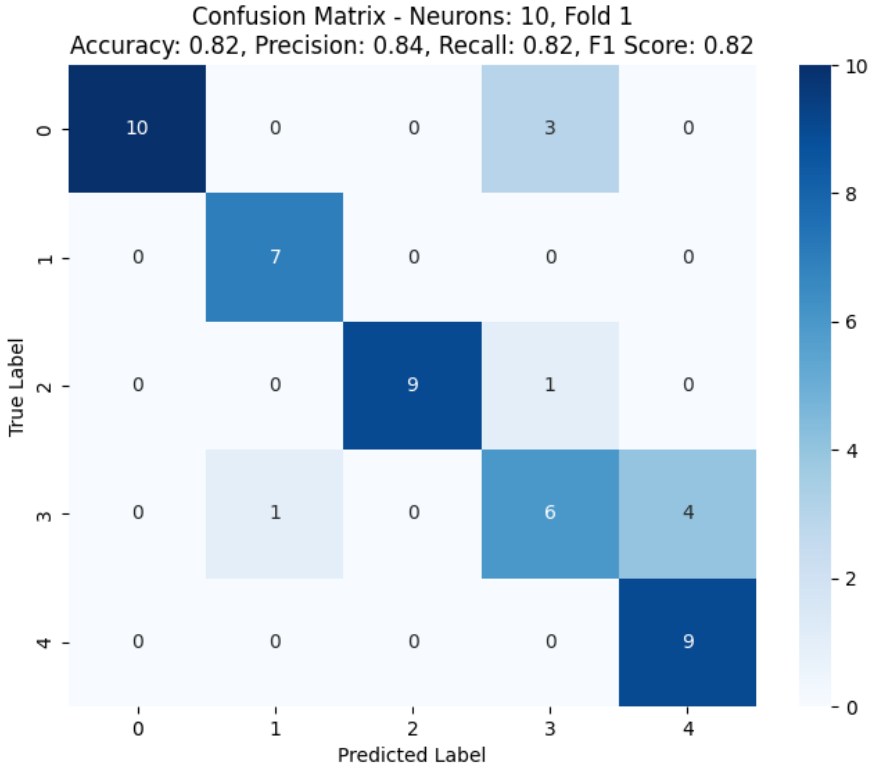

Confusion Matrix - Neurons: 5, Fold 1
Accuracy: 0.78, Precision: 0.80, Recall: 0.78, F1 Score: 0.77
True Label
Predicted Label
10 0 2 1 0
0 7 0 0 0
0 1 9 0 0
0 1 0 5 5
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10

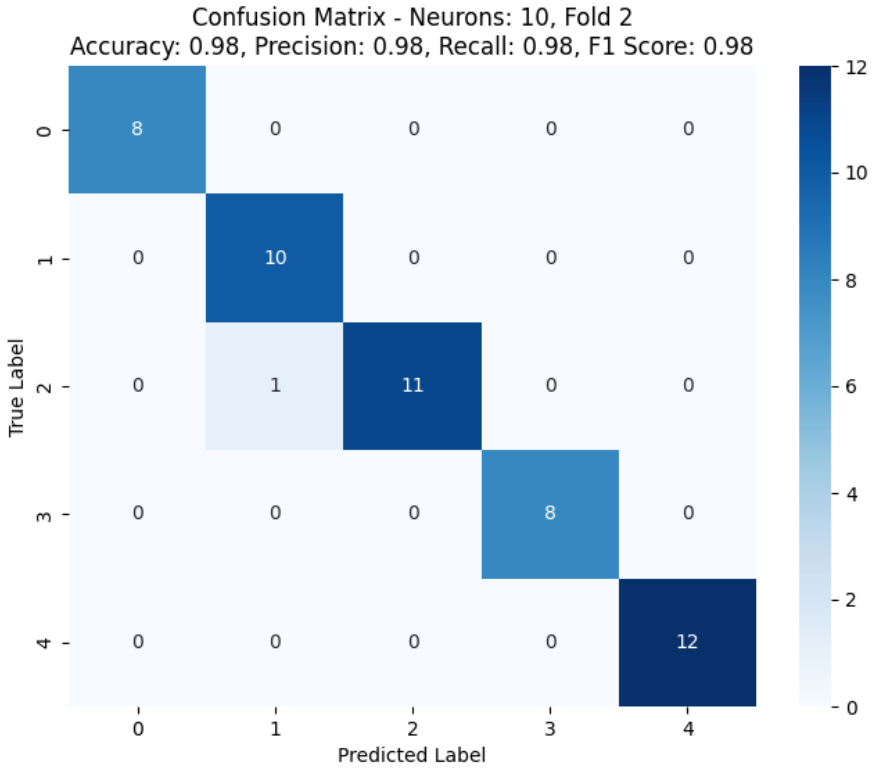

Confusion Matrix - Neurons: 5, Fold 2
Accuracy: 0.86, Precision: 0.89, Recall: 0.86, F1 Score: 0.85
True Label
Predicted Label
7 0 1 0 0
0 10 0 0 0
0 1 11 0 0
0 4 0 3 1
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

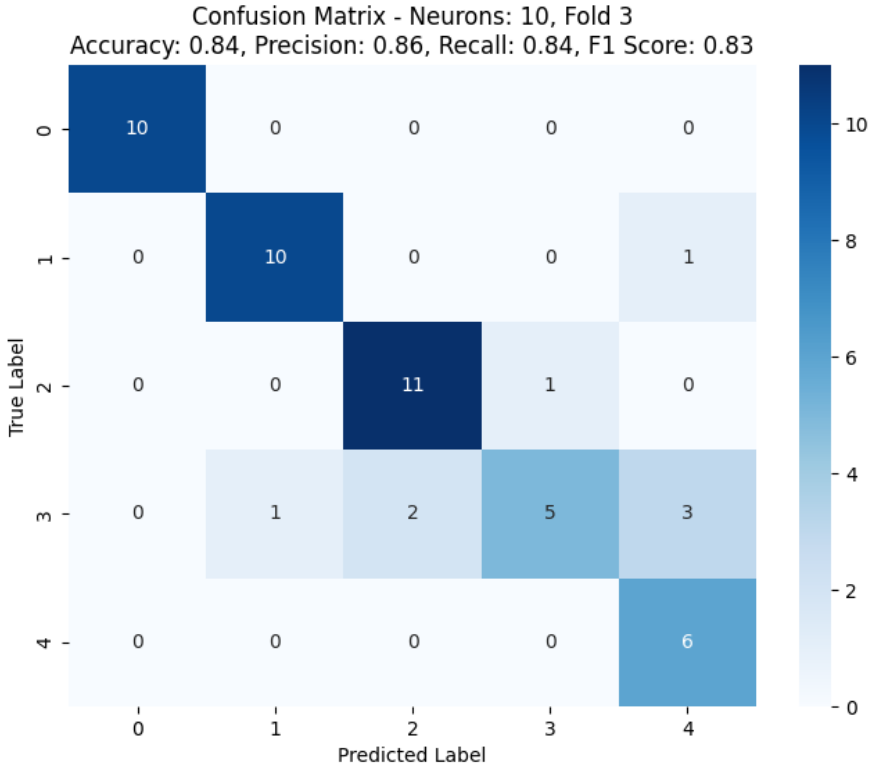

Confusion Matrix - Neurons: 5, Fold 3
Accuracy: 0.80, Precision: 0.80, Recall: 0.80, F1 Score: 0.80
True Label
Predicted Label
10 0 0 0 0
0 9 0 2 0
0 0 12 0 0
0 0 3 5 3
0 0 0 2 4
0 1 2 3 4
0 2 4 6 8 10 12

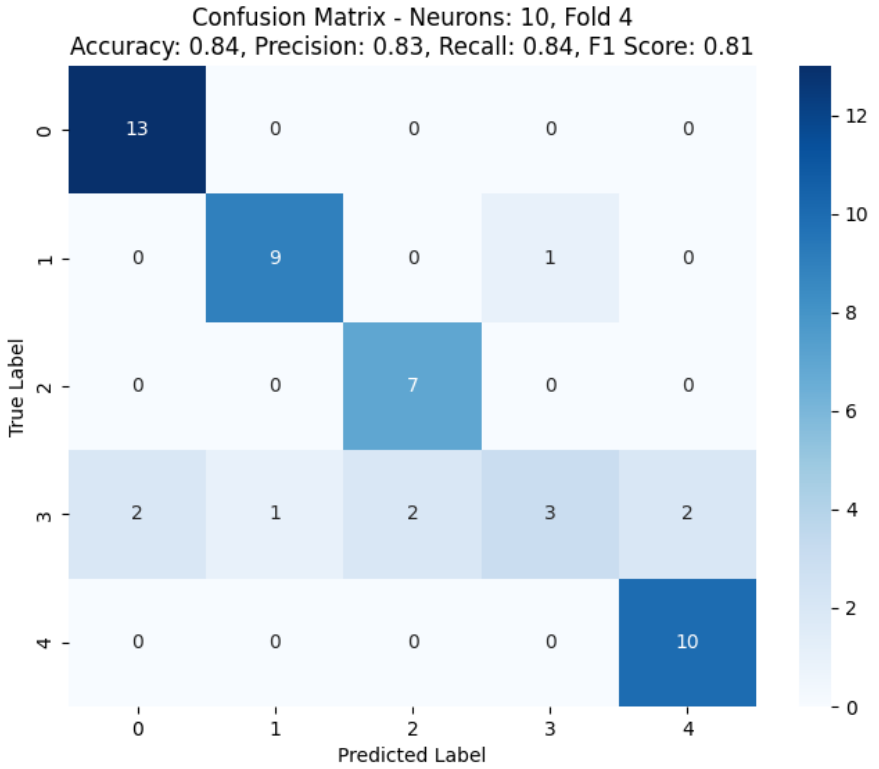

Confusion Matrix - Neurons: 5, Fold 4
Accuracy: 0.82, Precision: 0.80, Recall: 0.82, F1 Score: 0.80
True Label
Predicted Label
12 0 0 1 0
0 9 0 1 0
0 0 7 0 0
2 1 2 3 2
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

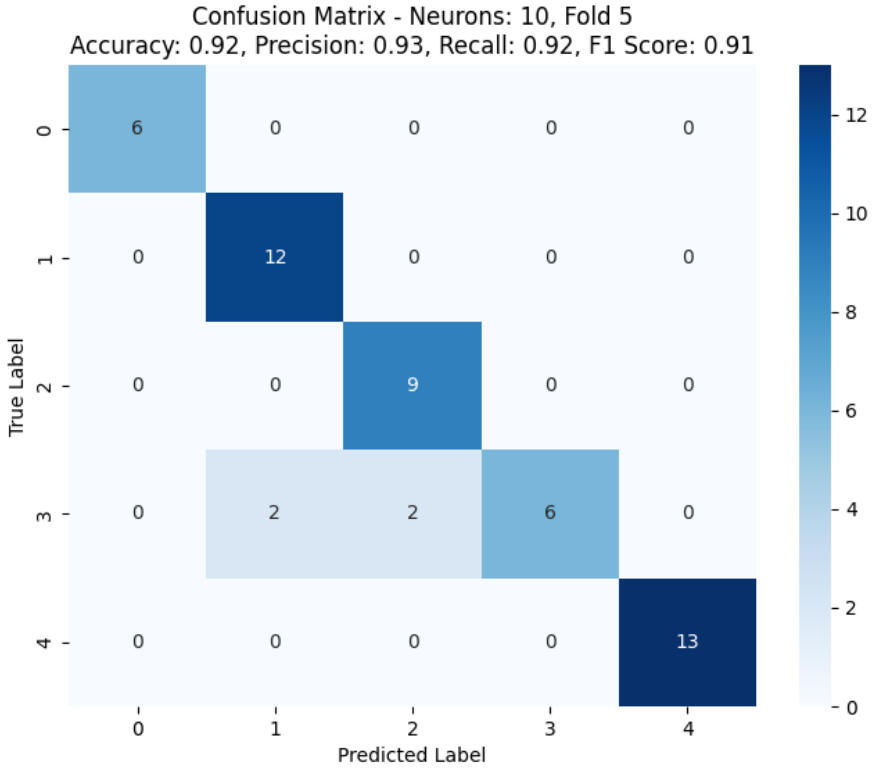

Confusion Matrix - Neurons: 5, Fold 5
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.89
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 1 3 5 1
0 0 0 0 13
0 1 2 3 4
0 2 4 6 8 10 12

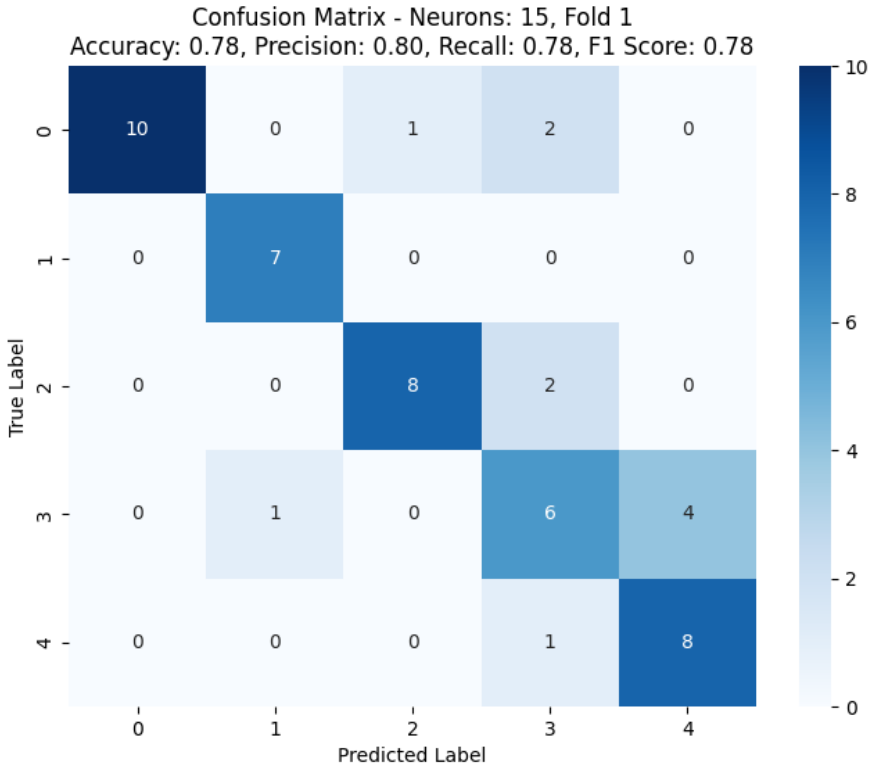

Confusion Matrix - Neurons: 10, Fold 1
Accuracy: 0.82, Precision: 0.84, Recall: 0.82, F1 Score: 0.82
True Label
Predicted Label
10 0 0 3 0
0 7 0 0 0
0 0 9 1 0
0 1 0 6 4
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10

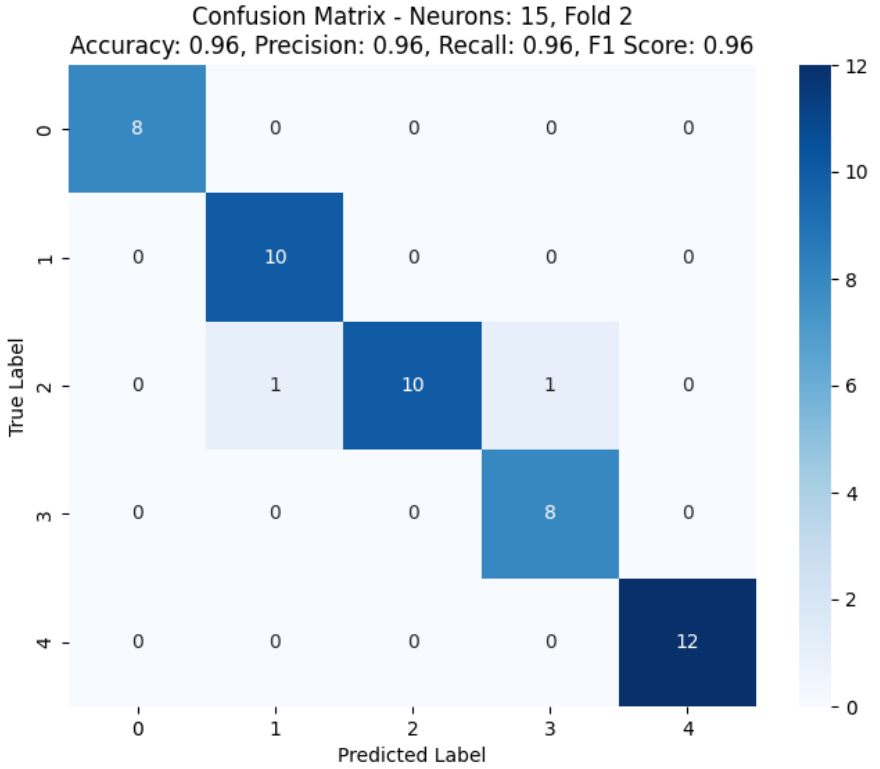

Confusion Matrix - Neurons: 10, Fold 2
Accuracy: 0.98, Precision: 0.98, Recall: 0.98, F1 Score: 0.98
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 1 11 0 0
0 0 0 8 0
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

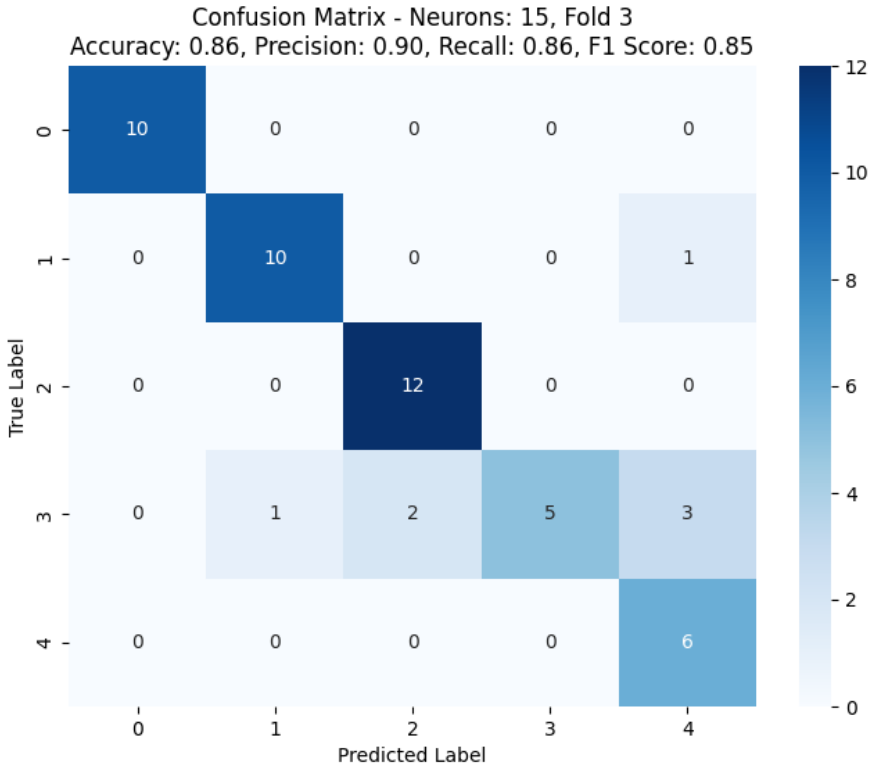

Confusion Matrix - Neurons: 10, Fold 3
Accuracy: 0.84, Precision: 0.86, Recall: 0.84, F1 Score: 0.83
True Label
Predicted Label
10 0 0 0 0
0 10 0 0 1
0 0 11 1 0
0 1 2 5 3
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10

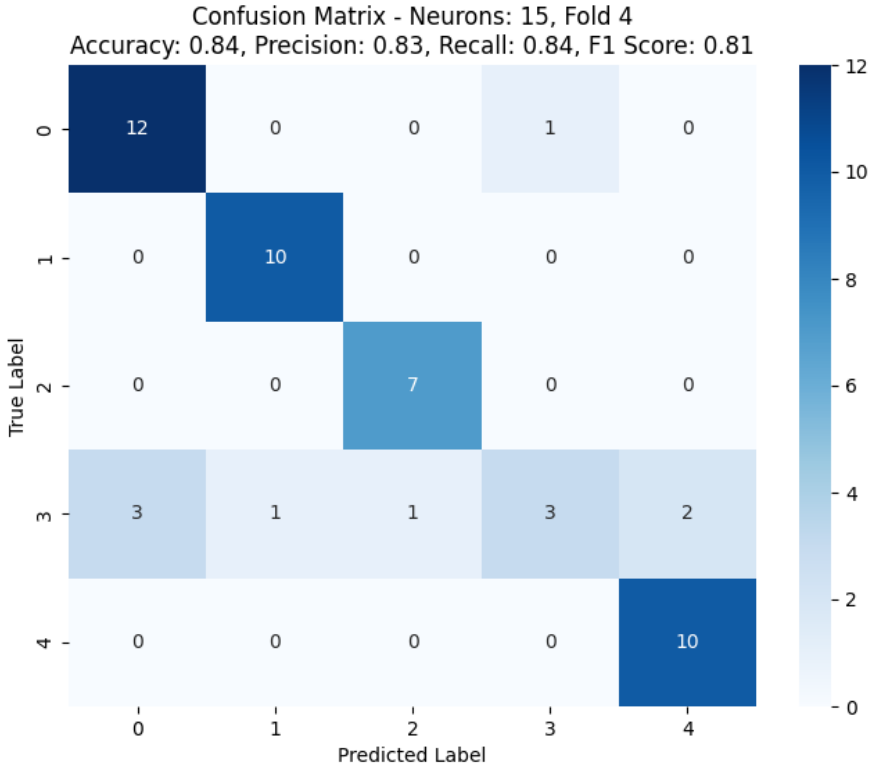

Confusion Matrix - Neurons: 10, Fold 4
Accuracy: 0.84, Precision: 0.83, Recall: 0.84, F1 Score: 0.81
True Label
Predicted Label
13 0 0 0 0
0 9 0 1 0
0 0 7 0 0
2 1 2 3 2
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

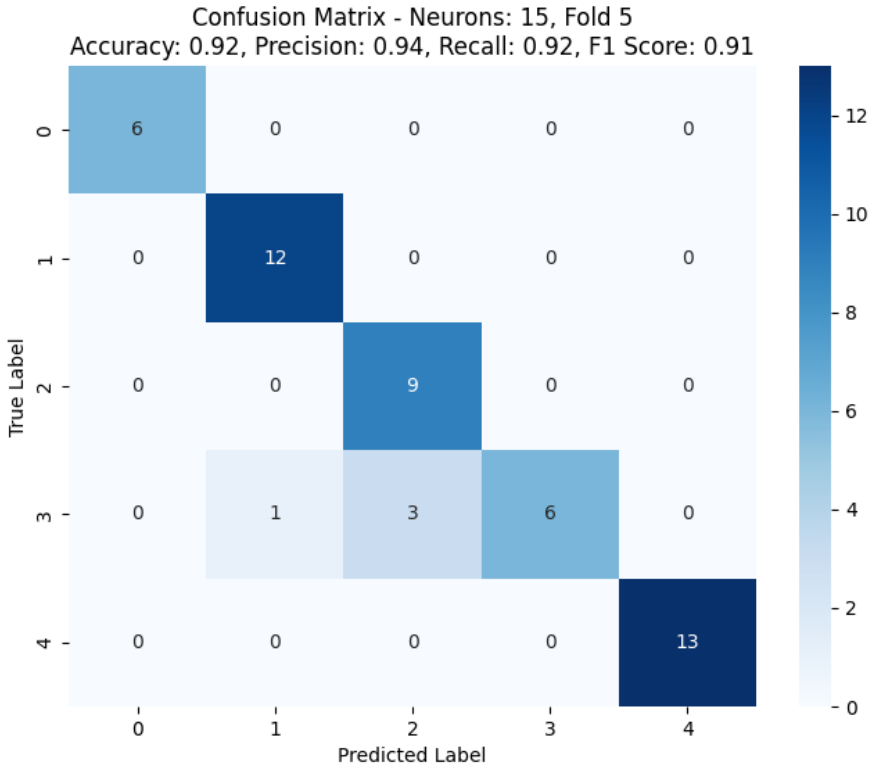

Confusion Matrix - Neurons: 10, Fold 5
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.91
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 2 2 6 0
0 0 0 0 13
0 1 2 3 4
0 2 4 6 8 10 12

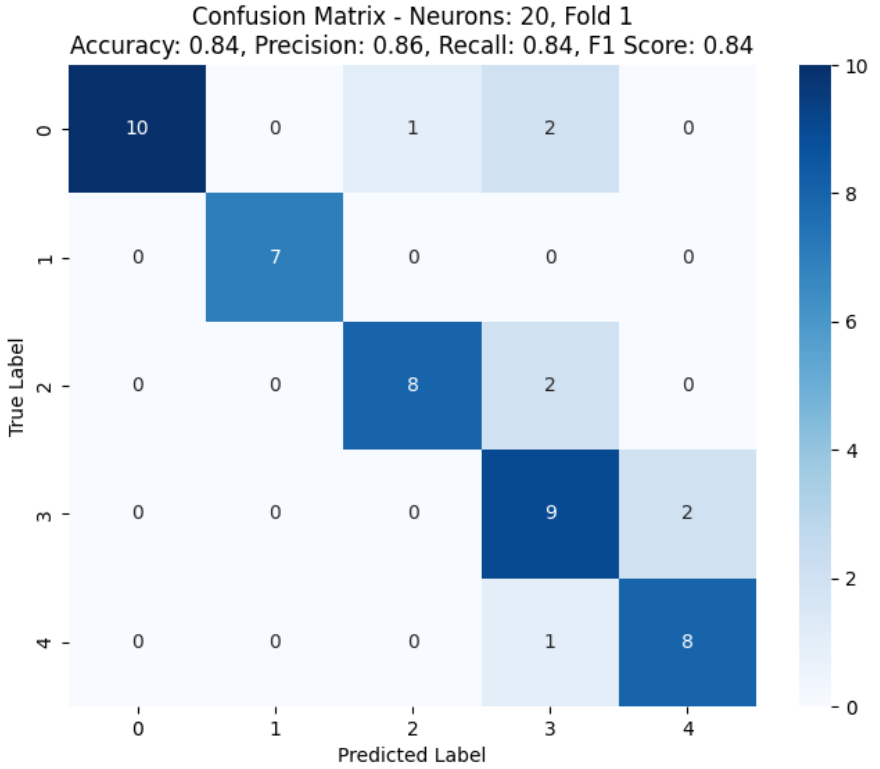

Confusion Matrix - Neurons: 15, Fold 1
Accuracy: 0.78, Precision: 0.80, Recall: 0.78, F1 Score: 0.78
True Label
Predicted Label
10 0 1 2 0
0 7 0 0 0
0 0 8 2 0
0 1 0 6 4
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10

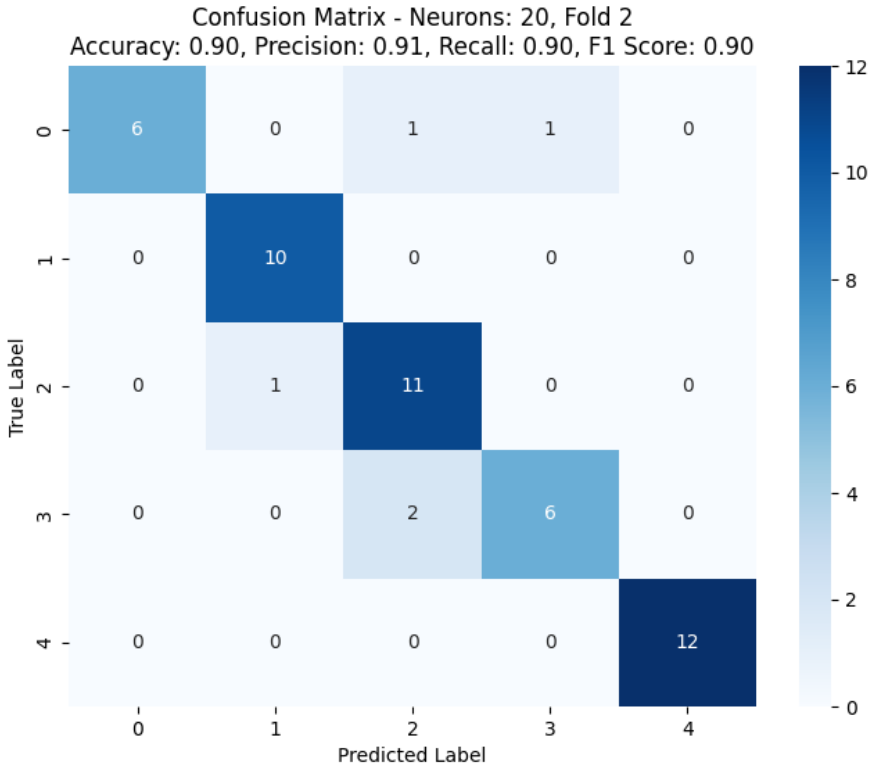

Confusion Matrix - Neurons: 15, Fold 2
Accuracy: 0.96, Precision: 0.96, Recall: 0.96, F1 Score: 0.96
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 1 10 1 0
0 0 0 8 0
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

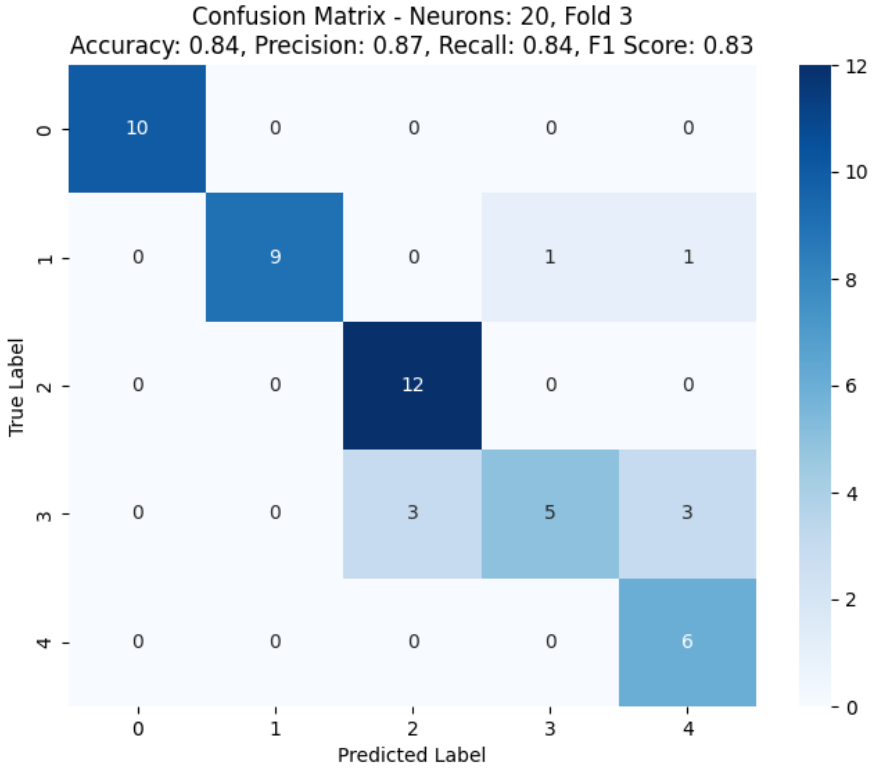

Confusion Matrix - Neurons: 15, Fold 3
Accuracy: 0.86, Precision: 0.90, Recall: 0.86, F1 Score: 0.85
True Label
Predicted Label
10 0 0 0 0
0 10 0 0 1
0 0 12 0 0
0 1 2 5 3
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12

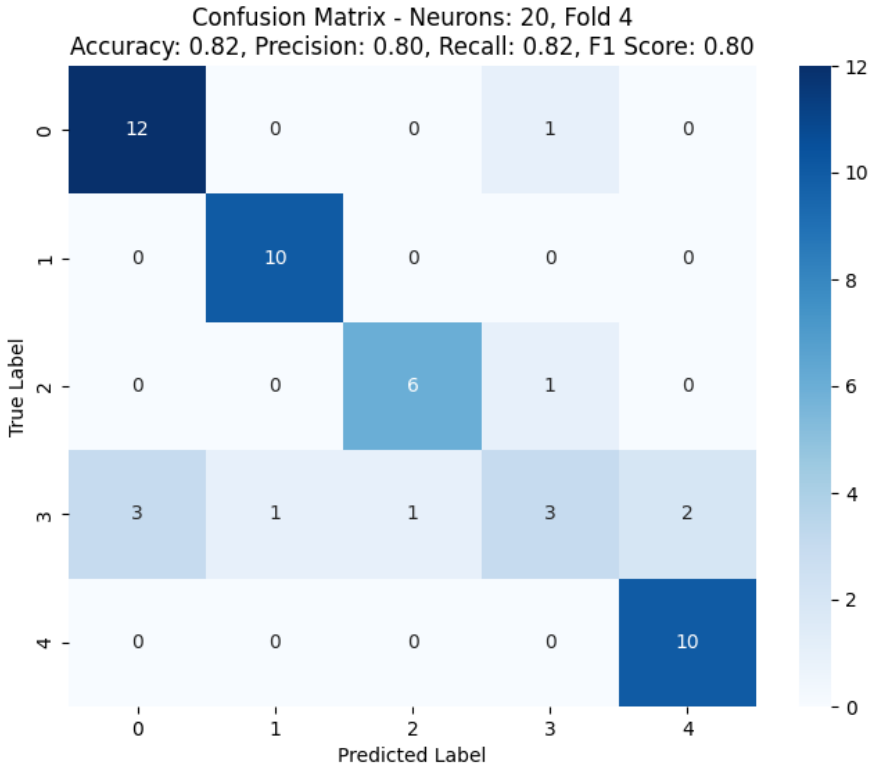

Confusion Matrix - Neurons: 15, Fold 4
Accuracy: 0.84, Precision: 0.83, Recall: 0.84, F1 Score: 0.81
True Label
Predicted Label
12 0 0 1 0
0 10 0 0 0
0 0 7 0 0
3 1 1 3 2
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

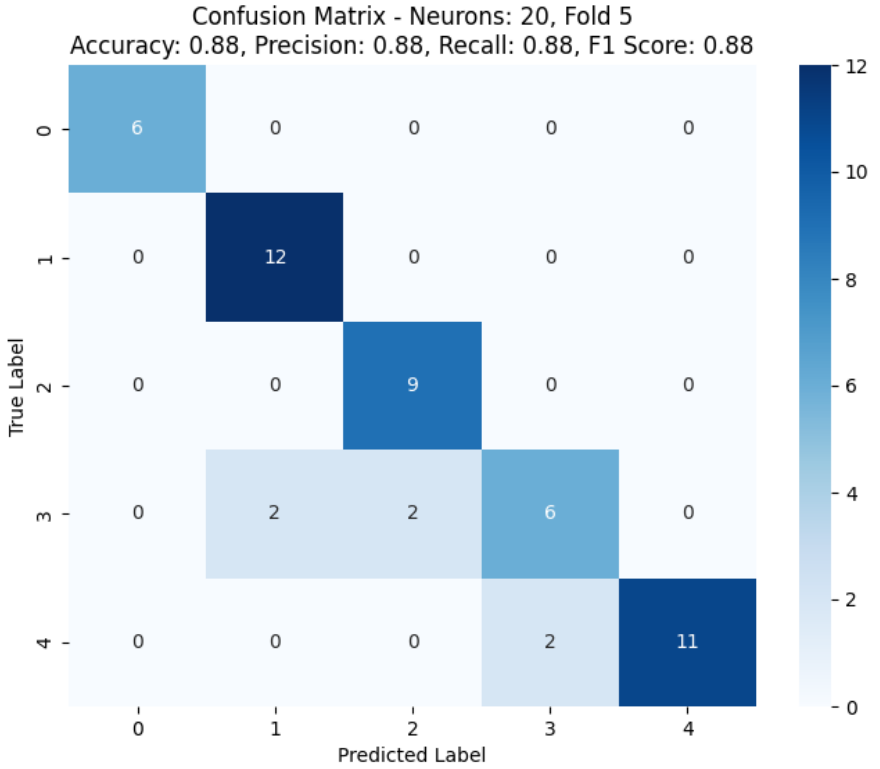

Confusion Matrix - Neurons: 15, Fold 5
Accuracy: 0.92, Precision: 0.94, Recall: 0.92, F1 Score: 0.91
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 1 3 6 0
0 0 0 0 13
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 1
Accuracy: 0.84, Precision: 0.86, Recall: 0.84, F1 Score: 0.84
True Label
Predicted Label
10 0 1 2 0
0 7 0 0 0
0 0 8 2 0
0 0 0 9 2
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10


Confusion Matrix - Neurons: 20, Fold 2
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
6 0 1 1 0
0 10 0 0 0
0 1 11 0 0
0 0 2 6 0
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 3
Accuracy: 0.84, Precision: 0.87, Recall: 0.84, F1 Score: 0.83
True Label
Predicted Label
10 0 0 0 0
0 9 0 1 1
0 0 12 0 0
0 0 3 5 3
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 4
Accuracy: 0.82, Precision: 0.80, Recall: 0.82, F1 Score: 0.80
True Label
Predicted Label
12 0 0 1 0
0 10 0 0 0
0 0 6 1 0
3 1 1 3 2
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 5
Accuracy: 0.88, Precision: 0.88, Recall: 0.88, F1 Score: 0.88
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 2 2 6 0
0 0 0 2 11
0 1 2 3 4
0 2 4 6 8 10 12

Bảng csv kết quả tổng quát

	A	B	C	D	E	F	G	H
1	neurons	fold	accuracy	precision	recall	f1	confusion_matrix	
2	5	1	0.78	0.800437	0.78	0.773147	[[10 0 2	
3	10	1	0.82	0.839115	0.82	0.819214	[[10 0 0	
4	15	1	0.78	0.800278	0.78	0.782318	[[10 0 1	
5	20	1	0.84	0.863206	0.84	0.844487	[[10 0 1	
6	5	2	0.86	0.894872	0.86	0.847006	[[7 0 1	
7	10	2	0.98	0.981818	0.98	0.980041	[[8 0 0	
8	15	2	0.96	0.96404	0.96	0.959246	[[8 0 0	
9	20	2	0.9	0.907532	0.9	0.898696	[[6 0 1	
10	5	3	0.8	0.802794	0.8	0.795179	[[10 0 0	
11	10	3	0.84	0.85841	0.84	0.830612	[[10 0 0	
12	15	3	0.86	0.897714	0.86	0.849038	[[10 0 0	
13	20	3	0.84	0.867333	0.84	0.830745	[[10 0 0	
14	5	4	0.82	0.798413	0.82	0.795429	[[12 0 0	
15	10	4	0.84	0.830889	0.84	0.811461	[[13 0 0	
16	15	4	0.84	0.828985	0.84	0.811532	[[12 0 0	
17	20	4	0.82	0.796485	0.82	0.795152	[[12 0 0	
18	5	5	0.9	0.917967	0.9	0.888389	[[6 0 0	
19	10	5	0.92	0.932987	0.92	0.913538	[[6 0 0	
20	15	5	0.92	0.936538	0.92	0.914686	[[6 0 0	
21	20	5	0.88	0.882987	0.88	0.875205	[[6 0 0	

Kết quả trung bình của các chỉ số cho từng số lượng neuron là:

- Neuron 5:
    - Accuracy: 0.832
    - Precision: 0.8429
    - Recall: 0.832
    - F1: 0.8198
- Neuron 10:
    - Accuracy: 0.880
    - Precision: 0.8886
    - Recall: 0.880
    - F1: 0.8710
- Neuron 15:
    - Accuracy: 0.872
    - Precision: 0.8855
    - Recall: 0.872
    - F1: 0.8634
- Neuron 20:
    - Accuracy: 0.856
    - Precision: 0.8635
    - Recall: 0.856
    - F1: 0.8489

Nhận xét:

1. Hiệu quả tăng từ 5 đến 10 neuron:
    - Khi tăng từ 5 lên 10 neuron, các chỉ số đều có sự cải thiện rõ rệt, đặc biệt là accuracy và precision. Điều này cho thấy mô hình hoạt động hiệu quả hơn khi có thêm neuron để học đặc trưng.
2. Hiệu quả gần đạt tối đa ở 10 neuron:
    - Với 10 neuron, các chỉ số đều đạt mức cao nhất hoặc gần cao nhất. Đây có thể là số lượng neuron tối ưu cho mô hình này trên tập dữ liệu hiện tại.

3. Giảm hiệu quả nhẹ khi tăng lên 15 và 20 neuron:
    - Khi tăng lên 15 và 20 neuron, các chỉ số bắt đầu giảm nhẹ, có thể do mô hình trở nên phức tạp hơn và có nguy cơ bị overfitting. Mức độ giảm không quá lớn nhưng cho thấy rằng việc tăng số neuron không phải lúc nào cũng cải thiện hiệu quả.

Kết luận: Việc tăng số lượng neuron giúp cải thiện hiệu quả mô hình, nhưng sau một ngưỡng (khoảng 10 neuron), sự cải thiện không còn rõ rệt và có thể làm giảm hiệu quả do phức tạp hóa mô hình

Dựa trên các kết quả từ hai phương pháp, phương pháp đầu tiên (ANN với HOG đặc trưng) và phương pháp thứ hai (ANN với Hu Moments đặc trưng), chúng ta có thể so sánh về ảnh hưởng của từng phương pháp đến hiệu quả của mô hình như sau:

1. Độ chính xác (Accuracy):

- Phương pháp ANN-HOG: Kết quả accuracy cao nhất là 0.968 khi sử dụng 20 neuron. Khi tăng số lượng neuron từ 5 đến 20, hiệu quả của mô hình tăng dần, đặc biệt là ở khoảng từ 10 neuron trở lên.
- Phương pháp ANN-HU: Kết quả accuracy cao nhất là 0.880 khi sử dụng 10 neuron, nhưng khi tăng thêm neuron (từ 15 đến 20), accuracy lại giảm nhẹ. Điều này cho thấy ANN-HU đạt hiệu quả tốt nhất ở mức neuron trung bình (10 neuron) và có xu hướng giảm nếu tăng quá nhiều neuron.

Nhận xét: ANN-HOG đạt độ chính xác cao hơn ANN-HU, đặc biệt khi số lượng neuron lớn. Điều này có thể do HOG đặc trưng tốt hơn trong việc nhận diện các chi tiết hình dạng và cấu trúc của đối tượng, đặc biệt là các đối tượng có nhiều đường biên và hình khối đặc trưng.

2. Precision:

- Phương pháp ANN-HOG: Precision cao nhất là 0.9693 với 20 neuron, cho thấy phương pháp này có khả năng phân loại chính xác cao hơn khi tăng neuron.
- Phương pháp ANN-HU: Precision cao nhất đạt 0.8886 với 10 neuron và cũng giảm nhẹ khi tăng neuron.

Nhận xét: Phương pháp ANN-HOG đạt precision cao hơn ANN-HU, cho thấy HOG đặc trưng giúp mô hình ít nhầm lẫn hơn khi phân loại, đặc biệt khi có nhiều neuron.

3. Recall:

- Phương pháp ANN-HOG: Recall cao nhất là 0.968 với 20 neuron, chứng tỏ mô hình có khả năng nhận diện tốt hơn các đối tượng thuộc các lớp mục tiêu khi sử dụng nhiều neuron.

- Phương pháp ANN-HU: Recall cao nhất là 0.880 với 10 neuron và cũng giảm nhẹ khi tăng neuron.

Nhận xét: Tương tự như precision, recall của ANN-HOG cao hơn ANN-HU, cho thấy rằng HOG giúp tăng khả năng nhận diện đầy đủ các mẫu thuộc các lớp mục tiêu.

4. F1-score:

- Phương pháp ANN-HOG: F1-score cao nhất là 0.9679 với 20 neuron, chứng tỏ mô hình có sự cân bằng tốt giữa precision và recall khi sử dụng HOG.
- Phương pháp ANN-HU: F1-score cao nhất là 0.8710 với 10 neuron và giảm nhẹ khi tăng neuron.

Nhận xét: F1-score của ANN-HOG cao hơn ANN-HU, đặc biệt là khi số neuron lớn, cho thấy HOG giúp mô hình đạt sự cân bằng giữa precision và recall tốt hơn Hu Moments.

5. Khả năng tối ưu với số lượng neuron:

- Phương pháp ANN-HOG: Phương pháp này đạt hiệu quả cao nhất ở mức 20 neuron, và hiệu quả tăng đều khi tăng số neuron.
- Phương pháp ANN-HU: Phương pháp này đạt hiệu quả tốt nhất ở mức 10 neuron, sau đó hiệu quả bắt đầu giảm nhẹ khi số neuron tăng. Điều này có thể do ANN-HU nhanh chóng đạt hiệu quả tối ưu và sau đó gặp overfitting khi mô hình phức tạp hơn.

Kết luận tổng quát:

Phương pháp ANN-HOG cho kết quả tốt hơn ANN-HU về tất cả các chỉ số (accuracy, precision, recall, và F1-score) và dường như tận dụng được nhiều hơn khi tăng số lượng neuron. Điều này cho thấy HOG là một đặc trưng phù hợp hơn cho việc nhận diện các đối tượng phức tạp, giúp mô hình ANN đạt hiệu quả cao hơn, đặc biệt với số lượng neuron lớn.

Ngược lại, ANN-HU có xu hướng đạt hiệu quả tối ưu sớm với số neuron trung bình (10 neuron) và có thể bị overfitting nếu tăng thêm. Điều này cho thấy Hu Moments có thể phù hợp hơn cho các bài toán đơn giản hoặc khi cần tiết kiệm tài nguyên tính toán, nhưng không hiệu quả bằng HOG cho các bài toán phân loại phức tạp hơn.

Số lượng neuron ẩn có ảnh hưởng đến hiệu suất của hệ thống ANN-Hu's Moments. Khi số neuron tăng, các chỉ số hiệu suất của hệ thống như accuracy, precision, recall, và f1 score cũng được cải thiện.

Hệ thống đạt hiệu suất tốt nhất khi số neuron ẩn đạt từ 15 đến 20. Tuy nhiên, với 15 neuron, hệ thống đã đạt đến hiệu quả gần tối ưu, và việc tăng lên 20 neuron không mang lại sự cải thiện đáng kể so với chi phí tính toán.

Tổng thể, hệ thống sử dụng Hu's Moments với ANN cho thấy khả năng phân loại tốt và ổn định, với các chỉ số đều dao động quanh mức cao, cho thấy mô hình có khả năng học tốt các đặc trưng của dữ liệu.

Bước 5: So sánh các đặc trưng. So sánh các thuật toán ML đã dùng. Nhận xét.

So sánh hiệu quả của hai đặc trưng Hu's Moments và HOG trong hệ thống phân loại sử dụng KNN:

So sánh chi tiết hai hệ thống:

K (Neighbors)	Hệ thống KNN-Hu's Moments	Hệ thống KNN-HOG
1	Accuracy: 0.92 Precision: 0.922 Recall: 0.919 F1: 0.915 Confusion Matrix: ít lỗi nhầm	Accuracy: 0.964 Precision: 0.969 Recall: 0.963 F1: 0.963 Confusion Matrix: hầu như không lỗi nhầm
3	Accuracy: 0.868 Precision: 0.866 Recall: 0.866 F1: 0.861 Confusion Matrix: bắt đầu có lỗi nhầm giữa một số lớp	Accuracy: 0.948 Precision: 0.955 Recall: 0.947 F1: 0.945 Confusion Matrix: ít lỗi nhầm
5	Accuracy: 0.84 Precision: 0.846 Recall: 0.837 F1: 0.828 Confusion Matrix: lỗi nhầm tăng, đặc biệt giữa các lớp gần nhau	Accuracy: 0.932 Precision: 0.940 Recall: 0.932 F1: 0.929 Confusion Matrix: vẫn giữ lỗi nhầm ít
7	Accuracy: 0.828 Precision: 0.830 Recall: 0.821 F1: 0.813 Confusion Matrix: lỗi nhầm giữa các lớp tăng nhiều	Accuracy: 0.916 Precision: 0.931 Recall: 0.916 F1: 0.912 Confusion Matrix: vẫn giữ lỗi nhầm ít

Nhận xét tổng quan:

1. Độ chính xác (Accuracy):
    - Hệ thống KNN-HOG duy trì độ chính xác cao hơn ở mọi giá trị của K so với KNN-Hu's Moments.

- Đặc biệt, ở K=1, KNN-HOG đạt độ chính xác 0.964, cao hơn đáng kể so với 0.92 của KNN-Hu's Moments.

2. Precision, Recall, và F1 Score:
    - KNN-HOG có precision, recall, và F1 score cao hơn so với KNN-Hu's Moments ở tất cả các giá trị K, đặc biệt là ở K=1.
    - Điều này chỉ ra rằng KNN-HOG phân loại chính xác hơn và có tỷ lệ dự đoán chính xác cho từng lớp tốt hơn.
3. Confusion Matrix:
    - Hệ thống KNN-HOG có rất ít lỗi nhầm lẫn giữa các lớp, ngay cả khi K tăng lên, trong khi KNN-Hu's Moments có số lượng lỗi nhầm giữa các lớp tăng lên đáng kể khi K tăng.
    - Hệ thống KNN-Hu's Moments dễ bị nhầm lẫn khi K lớn, cho thấy rằng đặc trưng Hu's Moments có thể không tối ưu trong việc phân biệt các lớp với nhiều láng giềng.
4. Hiệu quả với giá trị K thấp:
    - Cả hai hệ thống đều hoạt động tốt nhất khi K=1, nhưng hiệu suất của KNN-HOG vượt trội hơn. Khi K tăng, KNN-HOG vẫn duy trì hiệu quả tốt, trong khi KNN-Hu's Moments bị giảm mạnh về hiệu suất.

Kết luận:

- KNN-HOG vượt trội hơn KNN-Hu's Moments về cả độ chính xác và tính ổn định khi số lượng K thay đổi.
- KNN-Hu's Moments phù hợp hơn cho các bài toán đơn giản hơn, nhưng trong bài toán phân loại lá cây này, đặc trưng HOG với KNN mang lại hiệu quả cao hơn trong việc phân biệt các loại lá.
- Khuyến nghị: KNN-HOG là lựa chọn tốt hơn, đặc biệt với giá trị K=1, vì nó đạt được độ chính xác và độ phân giải lớp cao hơn, đồng thời ít gặp lỗi nhầm lẫn.

So sánh hiệu quả của hai đặc trưng Hu's Moments và HOG trong hệ thống phân loại sử dụng ANN:

1. Hu's Moments:

Số neuron ẩn từ 5 đến 20:

Accuracy: Trung bình dao động từ 0.832 đến 0.856.

Precision: Từ 0.8429 đến 0.8635, có sự ổn định và cải thiện nhẹ khi tăng số lượng neuron.

Recall: Từ 0.832 đến 0.856, cho thấy khả năng phát hiện tốt các mẫu đúng.

F1 score: Từ 0.8198 đến 0.8489, cũng tăng khi số lượng neuron tăng, thể hiện sự cân bằng giữa precision và recall.

Nhận xét:

Hu's Moments là một đặc trưng dựa trên hình dạng, giúp hệ thống phân biệt tốt giữa các loại mẫu với độ chính xác tương đối cao. Khi số lượng neuron tăng, hiệu suất của mô hình cải thiện dần nhưng giảm sau 15 neuron.

2. HOG (Histogram of Oriented Gradients):

Số neuron ẩn từ 5 đến 20:

Accuracy: Trung bình từ 0.836 đến 0.916. Hệ thống đạt hiệu suất tốt nhất ở mức 20 neuron với accuracy cao nhất là 0.916.

Precision: Dao động từ 0.7892 đến 0.9436, cho thấy khi số lượng neuron tăng, mô hình cải thiện rõ rệt khả năng nhận diện các mẫu chính xác.

Recall: Từ 0.836 đến 0.916, tăng khi tăng số neuron.

F1 score: Từ 0.799 đến 0.9087, tăng dần với số lượng neuron, nhưng sự tăng không quá lớn sau 20 neuron.

Nhận xét:

HOG dựa vào thông tin về các cạnh và hướng của gradient trong hình ảnh, giúp nhận diện tốt các mẫu có cấu trúc hình học phức tạp. Mô hình với đặc trưng HOG tăng đáng kể về precision và f1 score khi số lượng neuron tăng.

So sánh tổng quan:

Tiêu chí	Hu's Moments	HOG
Độ chính xác (Accuracy)	Ổn định từ 0.832 đến 0.856	Tăng dần từ 0.832 đến 0.880
Độ chính xác (Precision)	Từ 0.8429 đến 0.8635	Từ 0.7892 đến 0.8886

Tiêu chí	Hu's Moments	HOG
Khả năng phát hiện (Recall)	Từ 0.832 đến 0.856	Từ 0.832 đến 0.880
F1 Score	Từ 0.8198 đến 0.8489	Từ 0.799 đến 0.8710
Tính ổn định	Ổn định với số lượng neuron ít	Cải thiện khi số neuron tăng
Đặc điểm đặc trưng	Phù hợp với mẫu có hình dạng	Phù hợp với mẫu có cấu trúc hình học phức tạp

Kết luận:

- Hu's Moments cho thấy sự ổn định với hiệu suất vừa phải ngay cả khi số lượng neuron ẩn thấp. Nó phù hợp với các mẫu có đặc điểm về hình dạng rõ ràng. Khi tăng số lượng neuron, hiệu suất không cải thiện quá nhiều sau 15 neuron.
- HOG cần nhiều neuron hơn để đạt hiệu suất cao, đặc biệt là về precision và f1 score, phù hợp hơn với các bài toán có dữ liệu phức tạp về cấu trúc hình học. Khi tăng số neuron, hệ thống cải thiện đáng kể về các chỉ số.
- Tóm lại, Hu's Moments phù hợp với hệ thống đơn giản, hiệu quả ổn định, trong khi HOG cho thấy sự phát triển mạnh mẽ hơn khi số lượng neuron tăng, đặc biệt trong các bài toán phức tạp hơn.

CHƯƠNG 3: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

3.1. Đánh giá hệ thống đã xây dựng

Hệ thống phân loại lá cây được xây dựng với quy trình gồm ba bước chính, bao gồm thu thập và trích xuất đặc trưng, áp dụng mô hình học máy, và đánh giá hiệu suất. Dưới đây là đánh giá chi tiết cho từng bước:

1. Thu thập dữ liệu và trích xuất đặc trưng từ hình ảnh lá cây

Dữ liệu: Hệ thống sử dụng một tập hợp các hình ảnh lá cây thuộc nhiều loại khác nhau. Để đảm bảo độ đa dạng và tính chính xác của hệ thống, tập dữ liệu cần có đủ số lượng hình ảnh đại diện cho từng loại lá cây, đồng thời được chụp dưới các góc độ và điều kiện ánh sáng khác nhau.

Trích xuất đặc trưng:

Hu Moments: Đây là bộ đặc trưng giúp mô tả hình dạng tổng thể của lá cây thông qua các đặc điểm không biến đổi theo phép quay, dịch chuyển và co giãn. Hu Moments hiệu quả trong việc nhận diện các đặc trưng hình học tổng quát của lá.

Histogram of Oriented Gradients (HOG): HOG giúp nắm bắt chi tiết về đường biên và cấu trúc của lá cây, đặc biệt là các đặc trưng vi mô như vân lá và biên dạng mép lá. Điều này có thể bổ sung cho Hu Moments khi cần phân biệt các loại lá có cấu trúc biên phức tạp hơn.

Đánh giá: Sự kết hợp giữa Hu Moments và HOG giúp tăng cường khả năng nhận diện của hệ thống, do chúng bổ sung cho nhau về mặt đặc trưng. Tuy nhiên, việc lựa chọn giữa hai phương pháp này tùy thuộc vào loại lá và sự phức tạp của các đặc điểm hình học. Trong bài toán này, HOG cho kết quả tốt hơn, đặc biệt khi số lượng neuron tăng.

2. Mô hình học máy KNN và ANN cho phân loại

KNN (K-Nearest Neighbors):

KNN là mô hình đơn giản nhưng hiệu quả trong việc phân loại dựa trên khoảng cách đến các điểm láng giềng. KNN thích hợp cho các bài toán phân loại với đặc trưng hình học và không yêu cầu huấn luyện phức tạp.

Tuy nhiên, KNN thường bị ảnh hưởng bởi các điểm nhiễu và có thể chậm nếu dữ liệu lớn, do phải tính toán khoảng cách đến toàn bộ điểm dữ liệu trong quá trình phân loại.

ANN (Artificial Neural Network):

ANN có khả năng học đặc trưng phức tạp và đạt hiệu quả cao hơn trong các bài toán phân loại đòi hỏi sự phân biệt chi tiết. ANN cũng cho phép điều chỉnh số lượng neuron để tối ưu hiệu suất.

Kết quả cho thấy, ANN với HOG đặc trưng hoạt động tốt nhất khi có khoảng 15-20 neuron, đạt được độ chính xác cao nhất (xấp xỉ 0.916 với HOG). Điều này cho thấy ANN có thể tận dụng các đặc trưng chi tiết của HOG và tạo ra mô hình phân loại tốt hơn KNN.

Đánh giá: ANN tỏ ra ưu thế hơn KNN trong bài toán này, đặc biệt khi kết hợp với đặc trưng HOG. Tuy nhiên, ANN yêu cầu thời gian và tài nguyên tính toán lớn hơn, nên chỉ phù hợp khi có đủ tài nguyên.

3. Đánh giá hiệu suất qua cross-validation (5-fold)

Hệ thống sử dụng phương pháp cross-validation với 5-fold để đánh giá hiệu suất của mô hình. Đây là một kỹ thuật phổ biến để đánh giá độ chính xác của mô hình khi huấn luyện và kiểm thử trên các tập dữ liệu khác nhau, giúp giảm thiểu sai lệch do phân chia ngẫu nhiên của dữ liệu.

Các chỉ số đánh giá:

- Accuracy: Độ chính xác cao nhất đạt được là 88.0% với ANN và HOG đặc trưng. Điều này chứng tỏ hệ thống có khả năng phân loại khá tốt giữa các loại lá cây khác nhau.
- Precision: Precision đạt gần 88.9%, cho thấy mô hình ít nhầm lẫn khi dự đoán một loại lá cụ thể.
- Recall: Recall đạt mức 88.0%, chứng tỏ hệ thống có khả năng nhận diện hầu hết các mẫu thuộc từng loại lá, không bỏ sót nhiều mẫu.
- F1-score: F1-score xấp xỉ 87.1% cho phương pháp ANN-HOG với 20 neuron, thể hiện sự cân bằng giữa precision và recall, đảm bảo tính ổn định của mô hình.

Đánh giá: Cross-validation giúp đánh giá toàn diện mô hình và tránh overfitting. Kết quả từ cross-validation cho thấy hệ thống đạt hiệu suất cao nhất với ANN-HOG, giúp khẳng định độ tin cậy của mô hình trên tập dữ liệu tổng thể.

Kết luận tổng quát

Hệ thống phân loại lá cây sử dụng ANN với HOG đặc trưng đạt hiệu quả tốt nhất, với độ chính xác, precision, recall, và F1-score đều cao. Điều này cho thấy ANN-HOG phù hợp cho bài toán phân loại các loại lá cây phức tạp. Kết hợp giữa trích xuất đặc trưng mạnh (HOG) và mô hình học máy mạnh (ANN) đã giúp hệ thống đạt được hiệu quả phân loại cao.

Tuy nhiên, hệ thống có thể gặp khó khăn với các loại lá có hình dạng quá giống nhau, và chi phí tính toán cao hơn khi dùng ANN với số lượng neuron lớn.

3.2. Hướng phát triển

Mở rộng dữ liệu: Thu thập thêm dữ liệu về các loại lá cây khác, đặc biệt là những loại lá có thể gây hại cho con người (như lá độc). Điều này sẽ giúp cải thiện độ chính xác của hệ thống và làm cho nó có tính ứng dụng cao hơn.

Tối ưu hóa mô hình: Thử các mô hình học sâu (Deep Learning) như CNN (Convolutional Neural Network) để cải thiện độ chính xác và tốc độ phân loại, vì CNN rất mạnh trong việc phân tích hình ảnh.

Tích hợp với các ứng dụng thực tế: Xây dựng ứng dụng di động hoặc web cho phép người dùng tải ảnh lá cây lên và nhận biết loại lá đó. Điều này sẽ mang lại giá trị thực tiễn cho hệ thống.

Xử lý các yếu tố môi trường: Hệ thống có thể được phát triển thêm để nhận diện lá trong các điều kiện ánh sáng khác nhau, thay đổi nền, hoặc bị che khuất một phần. Điều này đòi hỏi việc cải thiện chất lượng dữ liệu hình ảnh hoặc áp dụng các kỹ thuật tiền xử lý tốt hơn.

Ứng dụng phân loại lá độc: Bổ sung thêm mô-đun nhận diện lá độc hại cho các ứng dụng liên quan đến an toàn sức khỏe, giáo dục về thực vật, hoặc bảo vệ môi trường.

PHỤ LỤC

Đường dẫn chứa code, bảng biểu, hình ảnh dữ liệu:
https://drive.google.com/drive/folders/1zzi-6hfklrKjzgn46NigHG4H9DXsnr0o?usp=sharing

Các file trong drive:

ANN: ANN_HOG + ANN_HU

- Chứa code phần ANN + HOG, các ma trận nhầm lẫn, file csv kết quả
- Chứa code phần ANN + Hu’s moments, các ma trận nhầm lẫn, file csv kết quả

Hog: chứa code trích xuất đặc trưng HOG ,ảnh xám, file csv dữ liệu chuẩn hóa std và chưa chuẩn hóa

Hu: chứa code trích xuất đặc trưng Hu’smoments ,ảnh xám, file csv dữ liệu chuẩn hóa std và chưa chuẩn hóa

KNN: KNN_HOG + KNN_HU

- Chứa code phần KNN + HOG, các ma trận nhầm lẫn, file csv kết quả
- Chứa code phần KNN + Hu’s moments, các ma trận nhầm lẫn, file csv kết quả

LEAF: chứa dữ liệu ảnh đầu vào của 3 thành viên trong nhóm và 2 bạn khác nhóm bao gồm ảnh gốc, ảnh nhị phân và ảnh xám.

README.txt

TÀI LIỆU THAM KHẢO

1. Hoàng Lê Uyên Thục, Nguyễn Văn Đức, Lê Thị Mỹ Hạnh. "So sánh đặc trưng moment Hu và biểu đồ gradient có hướng trong nhận dạng tự động hoa cảnh Việt Nam." Link to document
2. Hoàng Lê Uyên Thục. *Slides Trí Tuệ Nhân Tạo*.
3. Scikit-Learn Documentation. "KNeighborsClassifier." Available at: https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
4. W3Schools. "Python K-Nearest Neighbors (KNN)." Available at: https://www.w3schools.com/python/python_ml_knn.asp
5. GeeksforGeeks. "K-Nearest Neighbours (KNN)." Available at: https://www.geeksforgeeks.org/k-nearest-neighbours/
6. Viblo. "KNN (K-Nearest Neighbors)." Available at: https://viblo.asia/p/knn-k-nearest-neighbors-1-djeZ14ejKWz
7. GeeksforGeeks. "Artificial Neural Networks and Its Applications." Available at: https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
8. GeeksforGeeks. "Artificial Neural Network in TensorFlow." Available at: https://www.geeksforgeeks.org/artificial-neural-network-in-tensorflow/
9. Kaggle. "Introduction to ANN in TensorFlow." Available at: https://www.kaggle.com/code/srivignesh/introduction-to-ann-in-tensorflow
10. TensorFlow. "Beginner Quickstart." Available at: https://www.tensorflow.org/tutorials/quickstart/beginner
11. W3Schools. "Python ML Cross Validation." Available at: https://www.w3schools.com/python/python_ml_cross_validation.asp
12. Seaborn Documentation. "seaborn.heatmap." Available at: https://seaborn.pydata.org/generated/seaborn.heatmap.html