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	50	50	0

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)

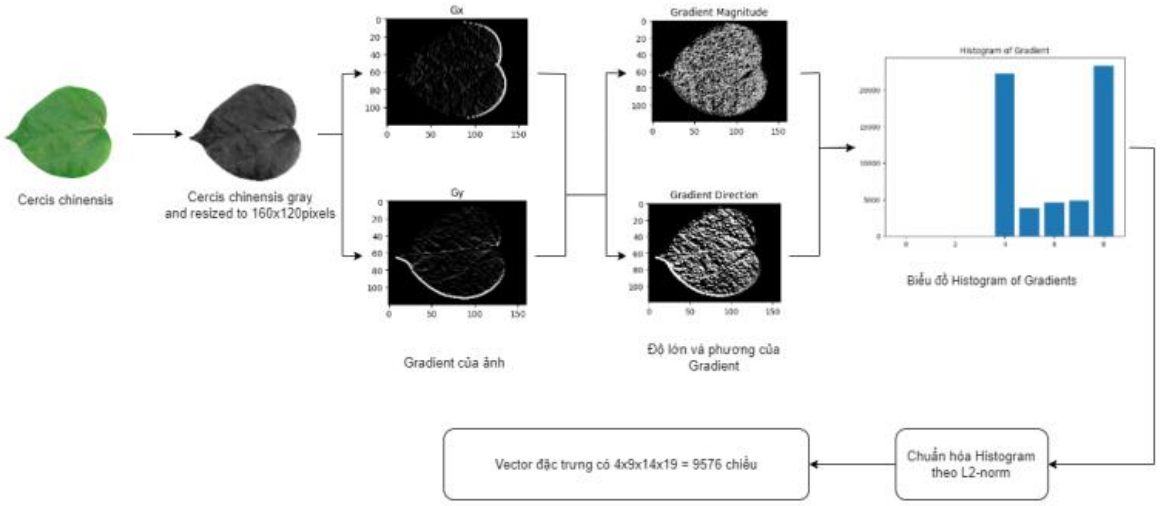
Phương pháp K-Nearest Neighbors (KNN) dựa trên nguyên lý so sánh khoảng cách giữa các điểm dữ liệu để thực hiện phân loại hoặc dự đoán. Khi thực hiện phân loại lá cây, KNN sẽ tìm kiếm K láng giềng gần nhất của một lá cần phân loại dựa trên các đặc trưng (ví dụ như Moment Hu, HOG) và sử dụng thông tin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

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.

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

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)

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_gray_folder='gray_images'):
    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

        # Lưu ảnh xám
        gray_image_path = os.path.join(output_gray_folder, f'gray_{i}')
        cv2.imwrite(gray_image_path, img_gray)

        # Trích xuất đặc trưng Hu Moments từ ảnh xám
        hu_features = extract_feature(img_gray)
        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)

# Đường dẫn ảnh đầu vào và lưu lại ảnh xám (DATALEAF)
output_gray_folder = r'E:/Downloads/DATA/Hu/gray_images'  # Thư mục lưu ảnh xám
la_chi, nhanlachi =
extract_hu_features_from_folder(      r"E:/Downloads/DATA/LEAF/THAIDUCTOAN/la_chi", 1, output_gray_folder=output_gray_folder)              #lachi
la_cham, nhanlacham =
extract_hu_features_from_folder(    r"E:/Downloads/DATA/LEAF/TRANTHANHKHOA/la_cham", 2, output_gray_folder=output_gray_folder)    #lacham
la_phong, nhanlaphong =
extract_hu_features_from_folder(  r"E:/Downloads/DATA/LEAF/MAIDUCKHIEM/la_phong", 3, output_gray_folder=output_gray_folder)              #laphong
la_tao, nhanlatao =
extract_hu_features_from_folder(      r"E:/Downloads/DATA/LEAF/NGOHUUMINH/latao", 4, output_gray_folder=output_gray_folder)                          #la tao
rau_muong, nhanraumuong = extract_hu_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/Hu/hutest_hsv/HUnhom11.csv')

# Chuẩn hóa dữ liệu Hu Moments
input_csv_path =    r'E:/Downloads/DATA/Hu/hutest_hsv/HUnhom11.csv'
output_csv_path =   r'E:/Downloads/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.

```
# Tải dataset từ file CSV

dataset =
pd.read_csv(r'E:/Downloads/DATA/hogtest_hsv/HOGhsv_nhom_5_standardized.csv')
```

	A	B	C	D	E	F	G	H
1	0	1	2	3	4	5	6	label
2	0.85698	1.362513	-0.32478	-0.82753	-0.2042	-0.5166	0.17439	1
3	-0.28477	0.14318	-0.56194	-0.76558	-0.20365	-0.47292	0.174442	1
4	0.886378	1.170872	-0.48093	-0.42258	-0.18675	-0.24105	0.186899	1
5	0.372353	0.924039	-0.41504	-0.37062	-0.17538	-0.15496	0.174637	1
6	0.275817	0.349895	-0.50904	-0.82448	-0.2042	-0.51575	0.17441	1
7	0.390469	0.882212	-0.5479	-0.37555	-0.1917	-0.06765	0.18278	1
8	0.406847	0.822216	1.136767	0.51374	0.236551	0.765374	-0.05336	1
9	0.110635	0.563058	-0.35545	-0.80559	-0.20397	-0.50215	0.173887	1
10	0.24316	0.368023	0.03987	0.572025	0.009946	0.576997	-0.18169	1
11	0.323819	0.648009	-0.33733	-0.777	-0.20489	-0.56644	0.172468	1
12	-0.1033	0.354833	-0.33828	-0.74283	-0.2025	-0.45641	0.17827	1
13	0.890461	1.553195	-0.37133	-0.24613	-0.16274	0.033375	0.210633	1
14	0.105544	0.35385	-0.53027	-0.54342	-0.21037	-0.77667	0.183503	1
15	0.765871	1.171303	-0.27612	-0.77767	-0.20555	-0.5617	0.173657	1
16	0.404632	1.02356	-0.1942	-0.66138	-0.2122	-0.68889	0.165104	1
17	0.32723	0.681221	-0.5712	-0.70026	-0.20351	-0.40786	0.176219	1
18	1.032275	1.318198	-0.39823	-0.77268	-0.20344	-0.56344	0.172591	1
19	0.25843	0.652071	-0.48463	-0.64064	-0.2029	-0.61418	0.164446	1
20	0.02609	0.471582	-0.46274	-0.77405	-0.20477	-0.56844	0.172969	1

145	-1.1028	-0.96055	-0.52389	-0.64386	-0.20384	-0.50482	0.182141	3
146	-0.90345	-0.60386	-0.26063	-0.31781	-0.23708	-0.86311	0.232816	3
147	-1.37113	-1.1636	-0.51065	-0.63638	-0.20295	-0.47153	0.183215	3
148	-1.14925	-0.84919	-0.29915	-0.30189	-0.21662	-0.6272	0.096689	3
149	-1.2907	-1.01254	-0.33316	-0.37958	-0.20131	-0.47603	0.114987	3
150	-0.9383	-0.5736	-0.08474	-0.04229	-0.23283	-0.7247	-0.01369	3
151	-0.94265	-0.49821	-0.06227	-0.13234	-0.27011	-0.93834	0.055477	3
152	0.944252	1.170983	0.011166	0.507944	-0.42933	-1.61182	-0.08293	4
153	0.604268	0.565044	0.085027	0.872388	-0.12404	0.491802	-0.52867	4
154	1.791088	2.462534	2.293293	2.567453	1.979521	3.136778	1.882187	4
155	1.790816	2.462646	2.293524	2.565741	1.97821	3.134773	-1.5312	4
156	0.604263	0.565038	0.08502	0.872378	-0.12404	0.491798	0.877436	4
157	0.603697	0.564182	0.084654	0.872413	-0.12416	0.491186	0.877301	4
158	0.596349	0.552902	0.07217	0.861116	-0.12691	0.481436	-0.51545	4
159	0.611918	0.925898	2.458356	1.948911	0.132189	0.579004	-2.95759	4
160	0.468658	0.393693	-0.33613	-0.27257	-0.24702	-0.70879	0.206656	4
161	0.910176	0.542734	1.897094	5.562695	5.401986	5.666669	-1.93501	4
162	0.942751	1.169978	0.01173	0.510492	-0.42969	-1.61098	0.433606	4
163	1.790024	2.463345	2.295189	2.560968	1.975134	3.129343	-1.52457	4

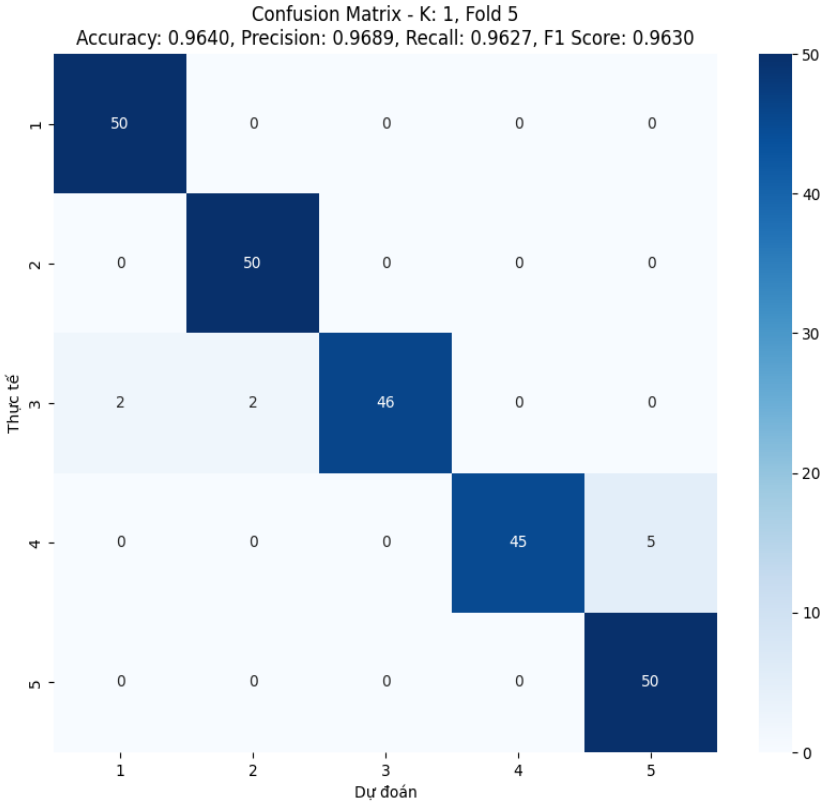
KNN-HOG:

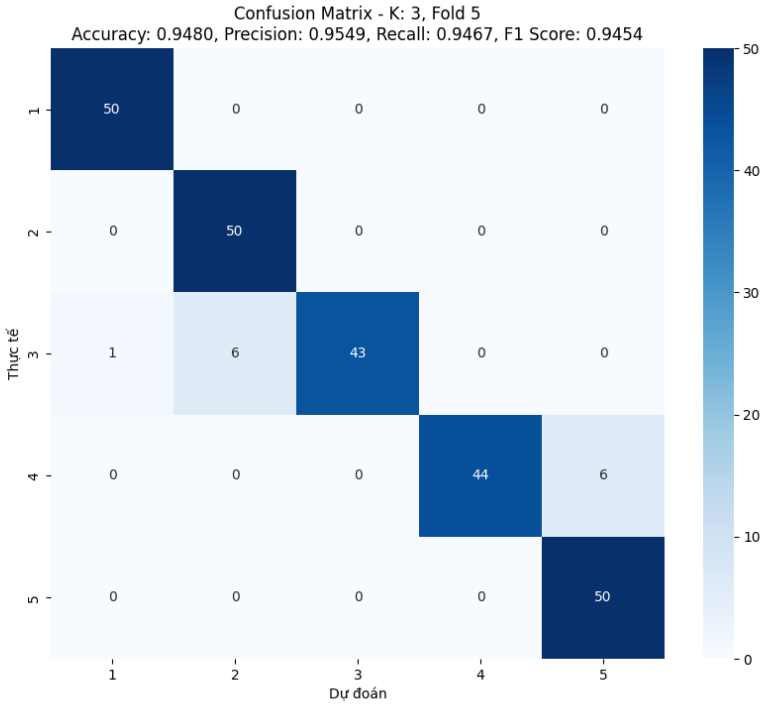
```
# Thiết lập mã hóa
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')


# 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)
```

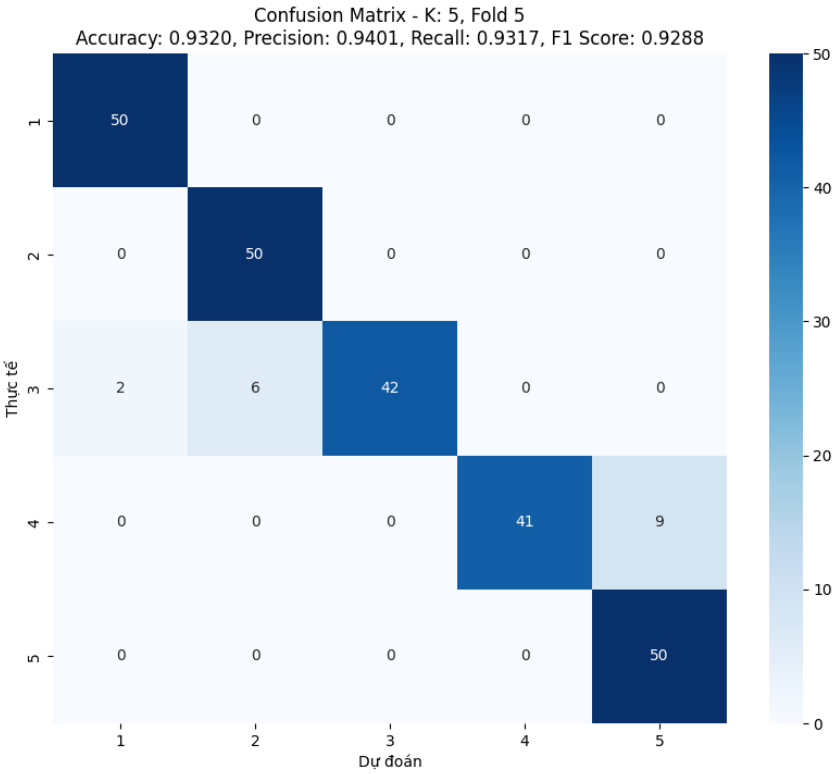
```
    # Dự đoán trên tập kiểm thử
    y_pred = classifier.predict(X_test)

    # 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}")
```

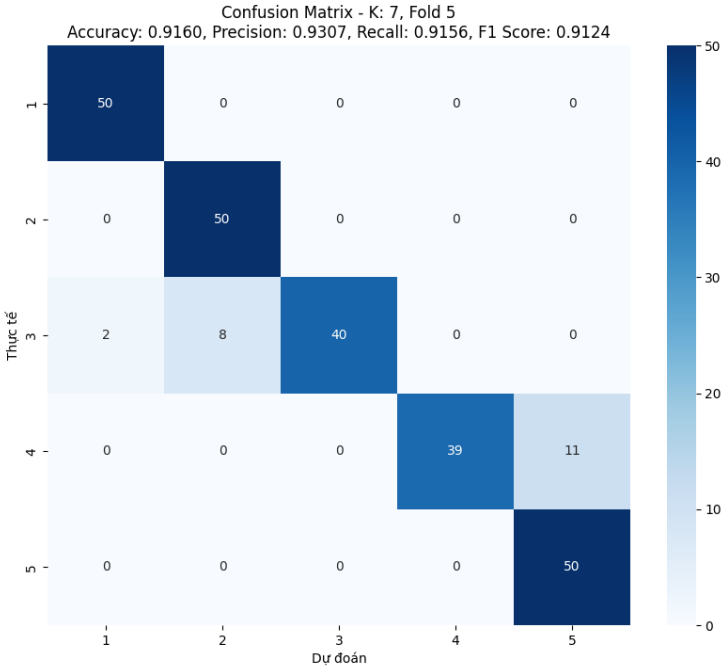
```
    print(f"K = {k} - Precision average: {precision_tb:.4f}")
    print(f"K = {k} - Recall average: {recall_tb:.4f}")
    print(f"K = {k} - F1 score average: {f1_tb:.4f}")

    # 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
0 10 20 30 40 50


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
0 10 20 30 40 50


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.

Hiệu quả tổng quan:

Hệ thống KNN-HOG cho thấy hiệu suất tốt khi sử dụng 5-fold cross-validation, với độ chính xác (accuracy) giảm dần khi giá trị của K tăng lên.

Điều này phản ánh một xu hướng phổ biến trong KNN: với K nhỏ, mô hình có thể phù hợp tốt hơn với các điểm dữ liệu gần nhất, nhưng khi K tăng, hệ thống bắt đầu tính toán theo nhiều điểm lân cận hơn, có thể dẫn đến việc làm mờ ranh giới giữa các lớp.

Sự ảnh hưởng của K đến hệ thống:

K = 1: Hệ thống đạt accuracy cao nhất (0.964) với f1 score là 0.963. Điều này cho thấy việc xét chỉ 1 lân cận gần nhất cho kết quả chính xác cao, nhưng có thể dễ dẫn đến overfitting, do nó chỉ phụ thuộc vào một điểm duy nhất trong không gian dữ liệu.

K = 3 và K = 5: Khi giá trị K tăng lên, độ chính xác giảm nhẹ (0.948 và 0.932 tương ứng). Tuy nhiên, điều này thường giúp mô hình trở nên ổn định hơn, giảm thiểu rủi ro overfitting. Mô hình trở nên tổng quát hơn khi xem xét thêm nhiều điểm lân cận, nhưng điều này cũng làm giảm độ chính xác một chút.

K = 7: Khi K tăng lên mức 7, accuracy giảm còn 0.916 và f1 score giảm xuống 0.912. Điều này có nghĩa là mô hình đang xem xét quá nhiều điểm lân cận, dẫn đến việc "làm mờ" ranh giới phân loại và có thể dẫn đến underfitting, khi mô hình không còn nhạy bén với các đặc trưng quan trọng.

Tính cân bằng giữa precision và recall:

Ở tất cả các giá trị của K, precision và recall duy trì mức độ tương đối đồng đều, nhưng nhìn chung precision cao hơn một chút so với recall khi K tăng lên. Điều này có nghĩa là mô hình vẫn giữ được khả năng phân loại đúng khá tốt, nhưng số lượng dự đoán sai cũng tăng dần khi K lớn hơn.

Kết luận:

K nhỏ (K = 1, 3) cho hiệu suất tốt hơn về độ chính xác, nhưng có thể dễ bị overfitting. K lớn hơn (K = 5, 7) giúp hệ thống trở nên ổn định hơn và tổng quát hóa tốt hơn, tuy nhiên sẽ giảm hiệu suất tổng thể.

Việc lựa chọn K cần phải cân nhắc kỹ lưỡng giữa độ chính xác và khả năng tổng quát hóa, phụ thuộc vào đặc tính của dữ liệu và yêu cầu của bài toán.

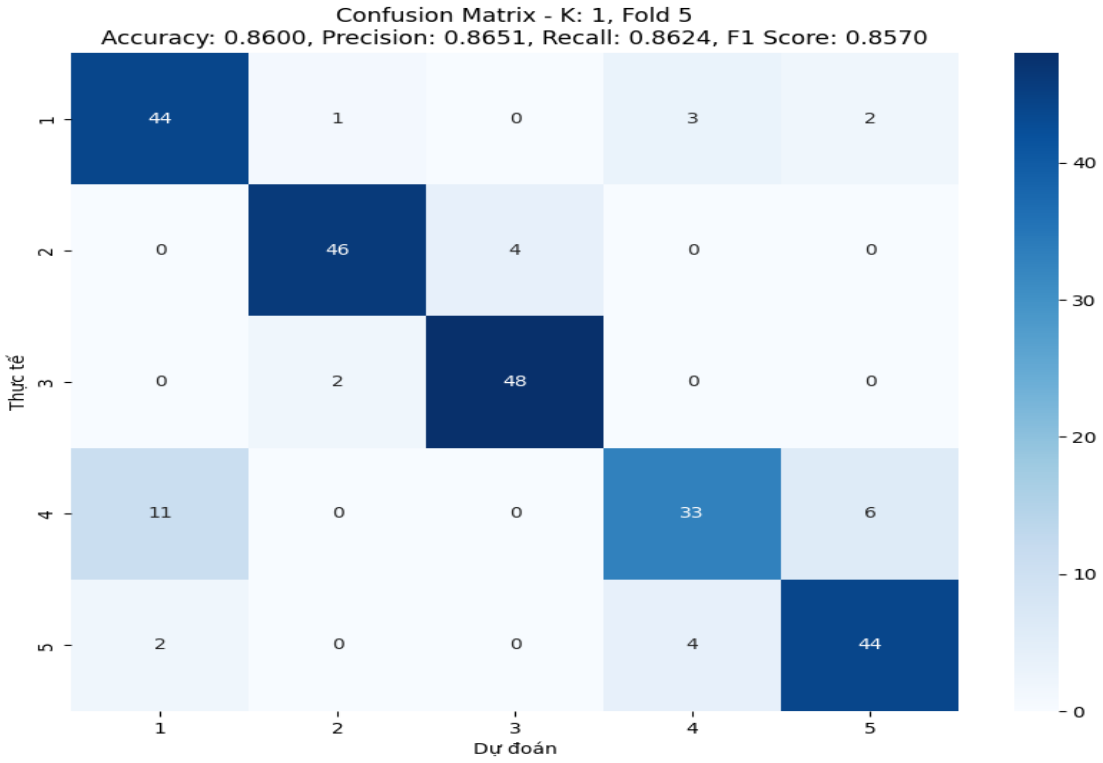
KNN-HU:

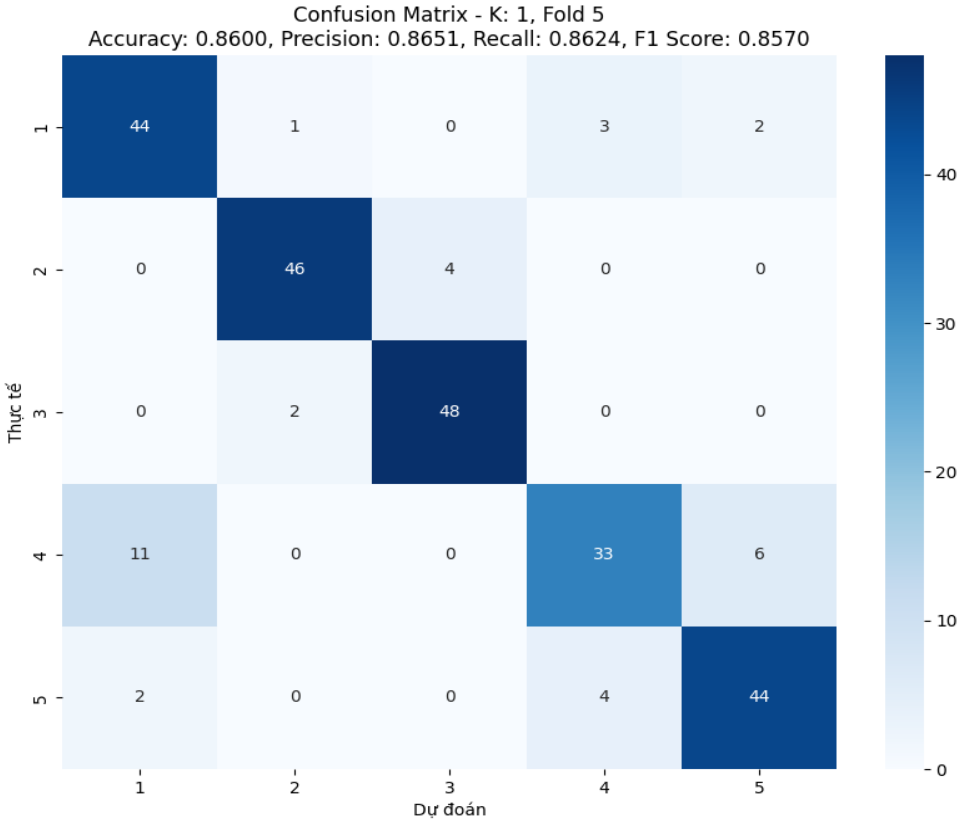
```
# Thiết lập mã hóa
sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')


# Bước 1: Tải dataset từ file CSV
dataset = pd.read_csv(r'E:/Downloads/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
```

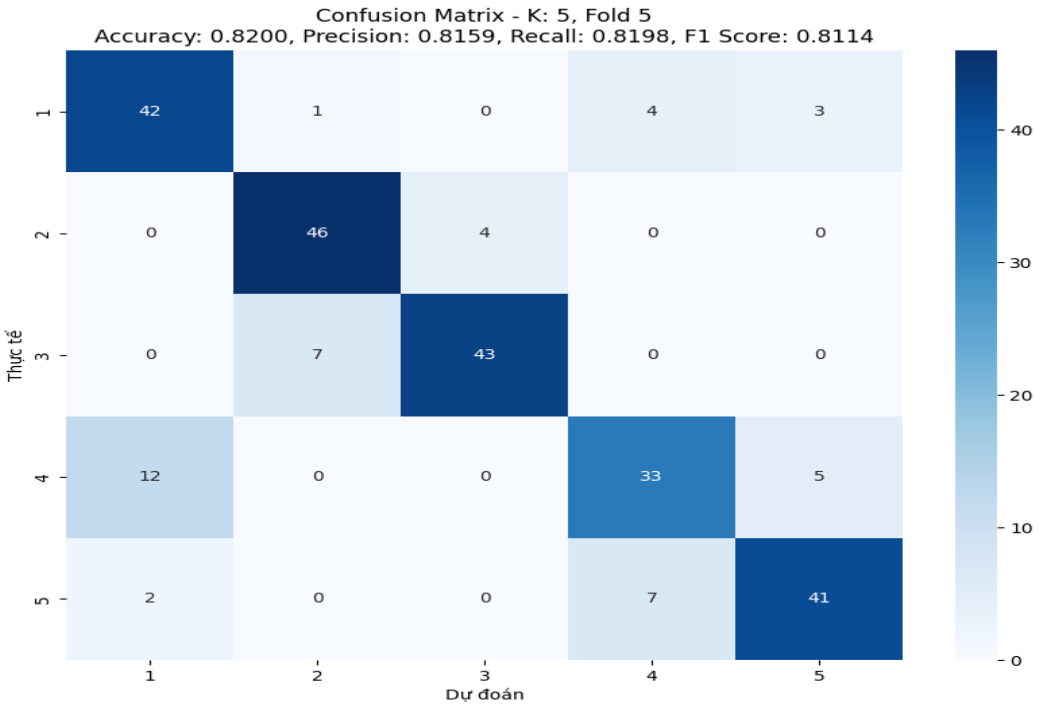
```
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # 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,
```

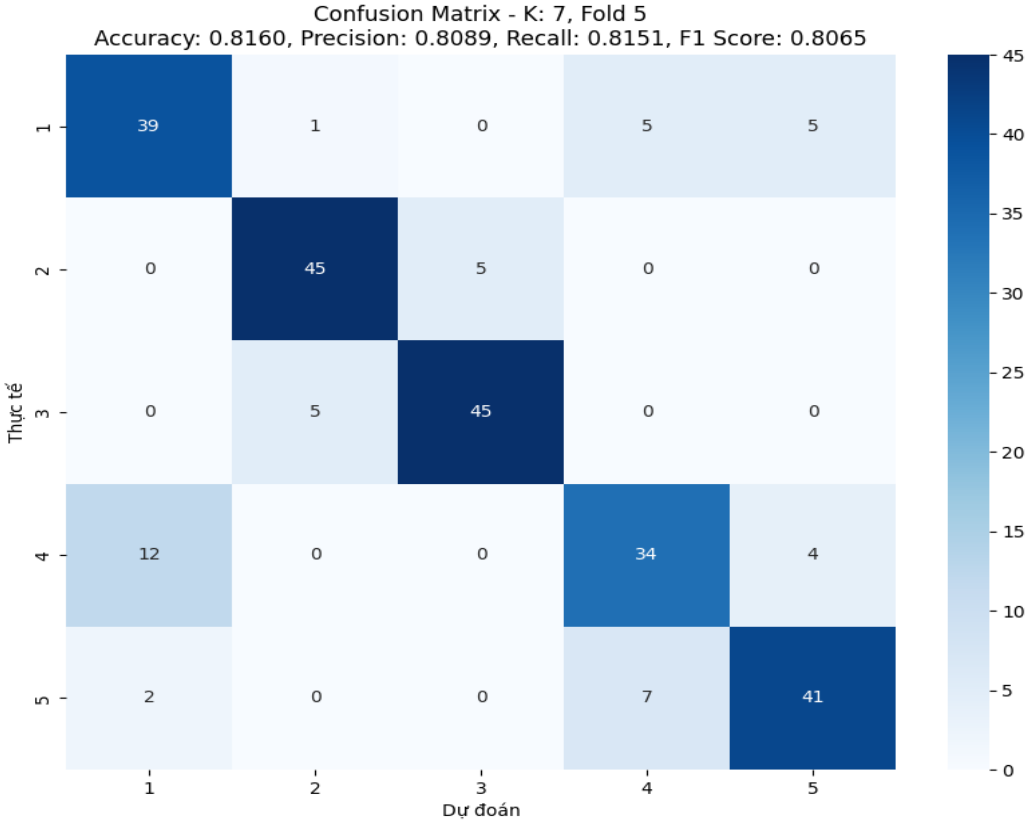
```
        'f1': f1_tb,
        'confusion_matrix': cm  # Lưu ma trận nhầm lẫn tổng
    })

    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: 1, Fold 5
Accuracy: 0.8600, Precision: 0.8651, Recall: 0.8624, F1 Score: 0.8570
Thực tế
Dự đoán
1
2
3
4
5
44 1 0 3 2
0 46 4 0 0
0 2 48 0 0
11 0 0 33 6
2 0 0 4 44
40
30
20
10
0


Confusion Matrix - K: 1, Fold 5
Accuracy: 0.8600, Precision: 0.8651, Recall: 0.8624, F1 Score: 0.8570
Thực tế
Dự đoán
1
2
3
4
5
44 1 0 3 2
0 46 4 0 0
0 2 48 0 0
11 0 0 33 6
2 0 0 4 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.

	K	fold	accuracy	precision	recall	f1	confusion_matrix
2	1	5	0.86	0.865087	0.862444	0.856962	[[44. 1. 0.
3	3	5	0.844	0.840061	0.843656	0.836406	[[42. 1. 0.
4	5	5	0.82	0.815892	0.819801	0.811387	[[42. 1. 0.
5	7	5	0.816	0.808889	0.815057	0.806509	[[39. 1. 0.

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

Hiệu quả tổng quan:

Hệ thống KNN sử dụng Hu's moments cho thấy độ chính xác (accuracy) giảm dần khi giá trị của K tăng lên, tương tự như kết quả của hệ thống KNN-HOG.

Mặc dù không đạt độ chính xác cao bằng KNN-HOG, mô hình này vẫn duy trì được khả năng phân loại tốt và sự ổn định qua các giá trị K khác nhau.

Sự ảnh hưởng của K đến hệ thống:

K = 1: Với K nhỏ nhất, mô hình đạt accuracy cao nhất (0.86) và f1 score là 0.857. Điều này cho thấy với K=1, việc chỉ xem xét 1 điểm lân cận có thể giúp hệ thống phù hợp với dữ liệu tốt hơn, nhưng cũng có khả năng dẫn đến overfitting, vì nó chỉ phụ thuộc vào một điểm duy nhất.

K = 3: Khi giá trị K tăng lên 3, độ chính xác giảm nhẹ xuống 0.844, nhưng mô hình có thể trở nên ổn định hơn, mặc dù kết quả vẫn thấp hơn so với K = 1.

K = 5 và K = 7: Khi K tiếp tục tăng lên 5 và 7, accuracy giảm còn 0.82 và 0.816, cho thấy mô hình bắt đầu bị ảnh hưởng bởi việc xem xét quá nhiều điểm lân cận, làm giảm khả năng phân loại chính xác và có nguy cơ underfitting khi mô hình không còn đủ nhạy bén với các đặc trưng.

Tính cân bằng giữa precision và recall:

Ở mọi giá trị K, precision và recall duy trì tương đối ổn định và cân bằng. Tuy nhiên, chúng có xu hướng giảm dần khi K tăng lên. Điều này phản ánh rằng với K lớn hơn, mô hình có xu hướng giảm khả năng phân loại chính xác (precision) và cũng khó khăn hơn trong việc nhận dạng đúng các mẫu của từng lớp (recall).

Kết luận:

K nhỏ (K = 1, 3) giúp hệ thống có độ chính xác cao hơn nhưng dễ bị overfitting do quá phụ thuộc vào các điểm dữ liệu lân cận. Trong khi đó, K lớn hơn (K = 5, 7) giúp mô hình ổn định và tổng quát hơn, nhưng đồng thời làm giảm độ chính xác do hiện tượng underfitting.

So với KNN-HOG, hệ thống KNN với Hu's moments cho hiệu suất thấp hơn ở tất cả các giá trị K, cho thấy rằng Hu's moments có thể không phù hợp bằng HOG cho bài toán phân loại này.

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.

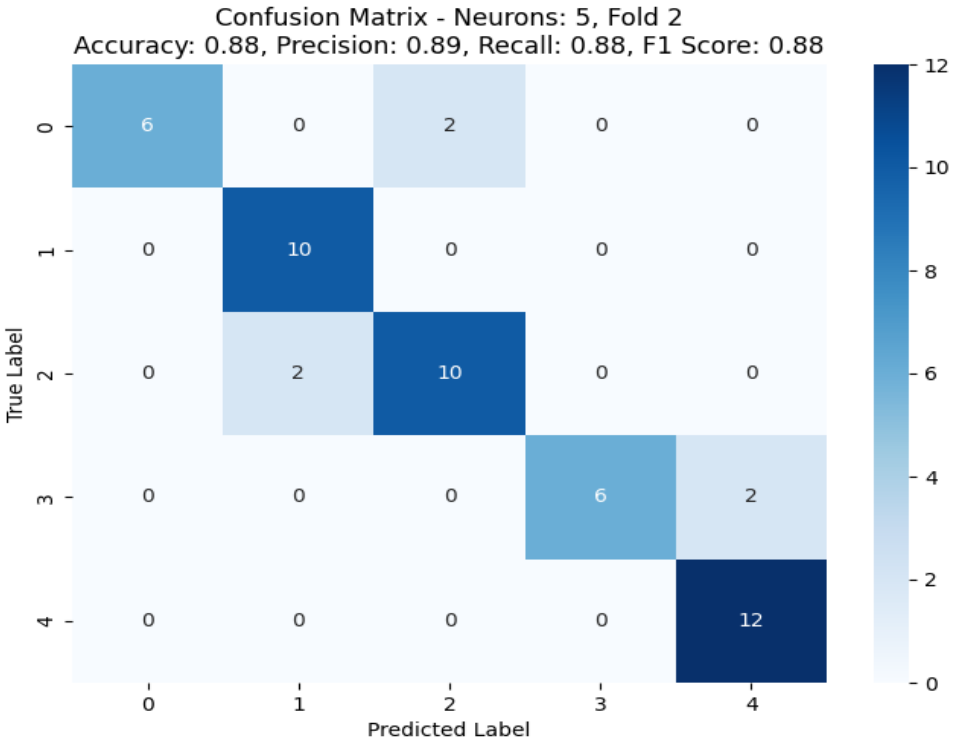
ANN-HOG:

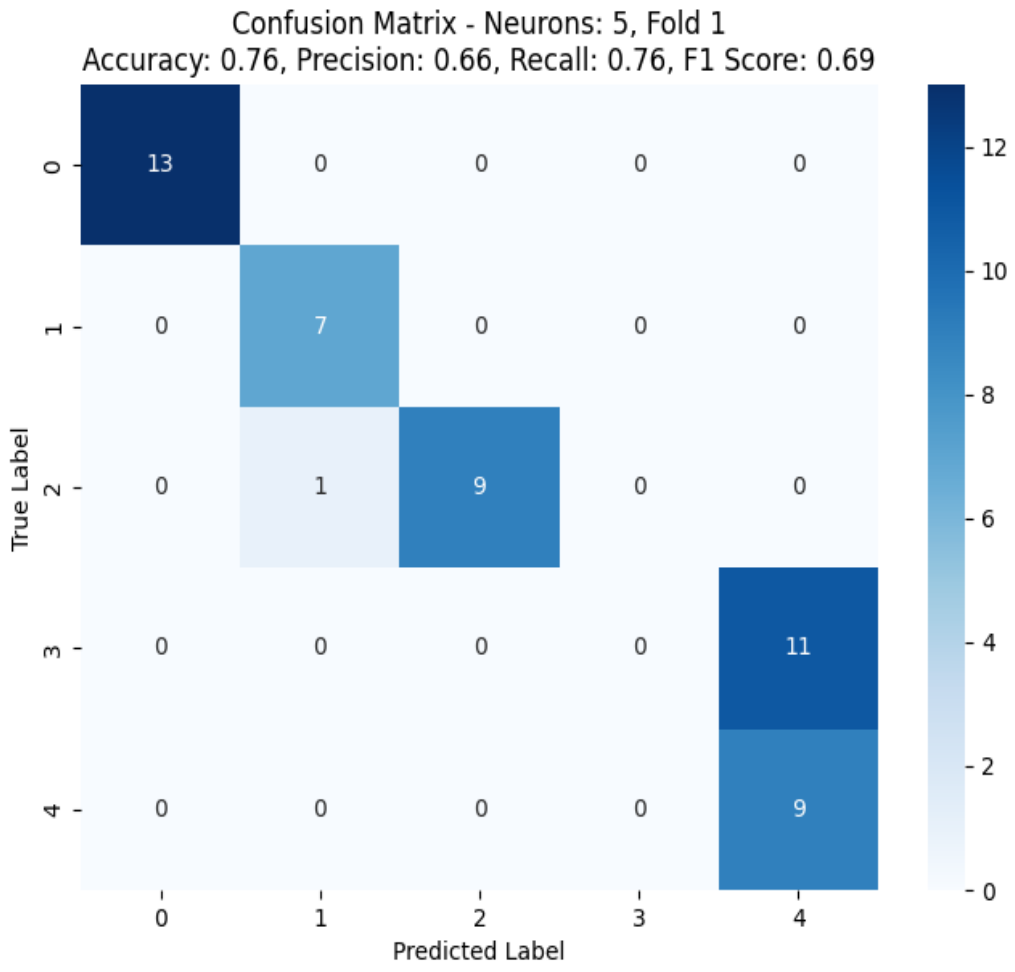
```
# Bước 1: Tải dữ liệu từ file CSV
data = pd.read_csv(r'E:/Downloads/DATA/Hog/hogtest_hsv/HOGnhom11Std.csv')

# 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'])
```

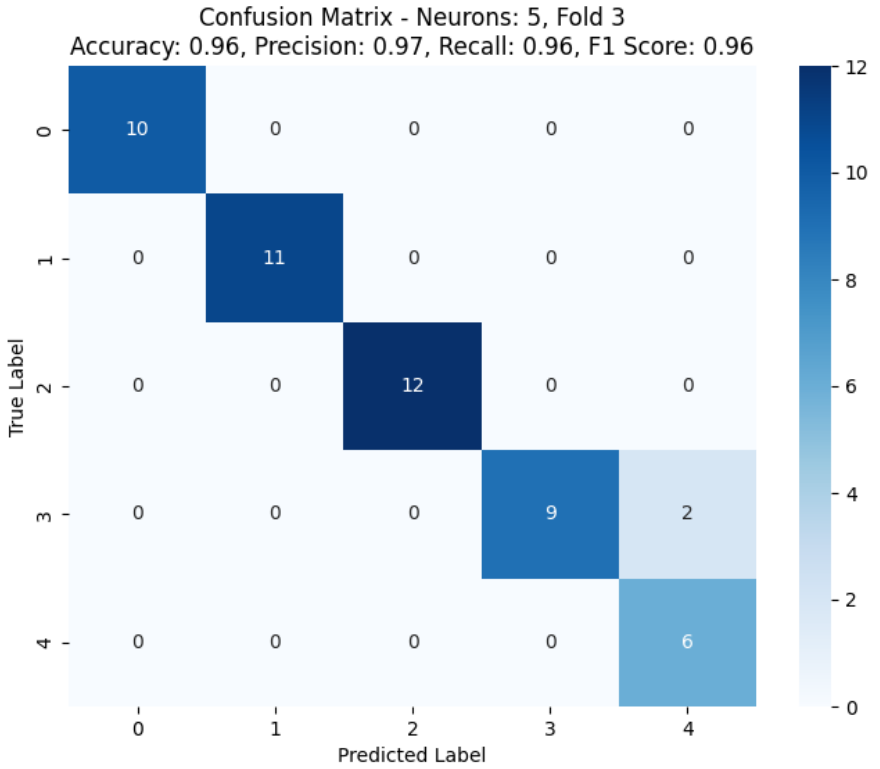
```
        # Huấn luyện mô hình
        model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)

        # 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)
```

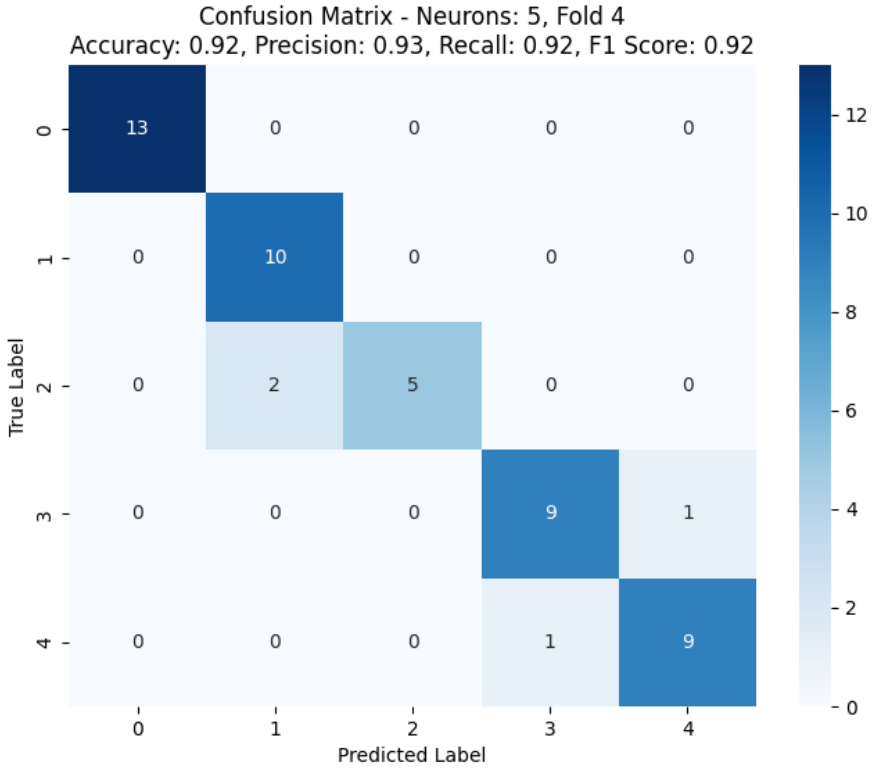
Kết quả:

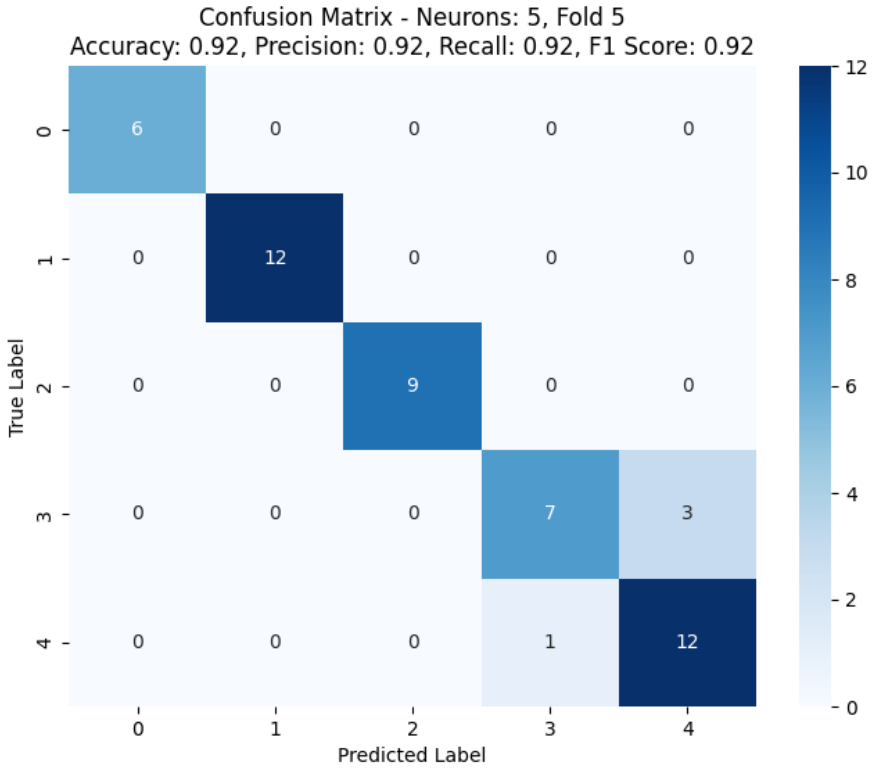
Ma trận nhầm lẫn:

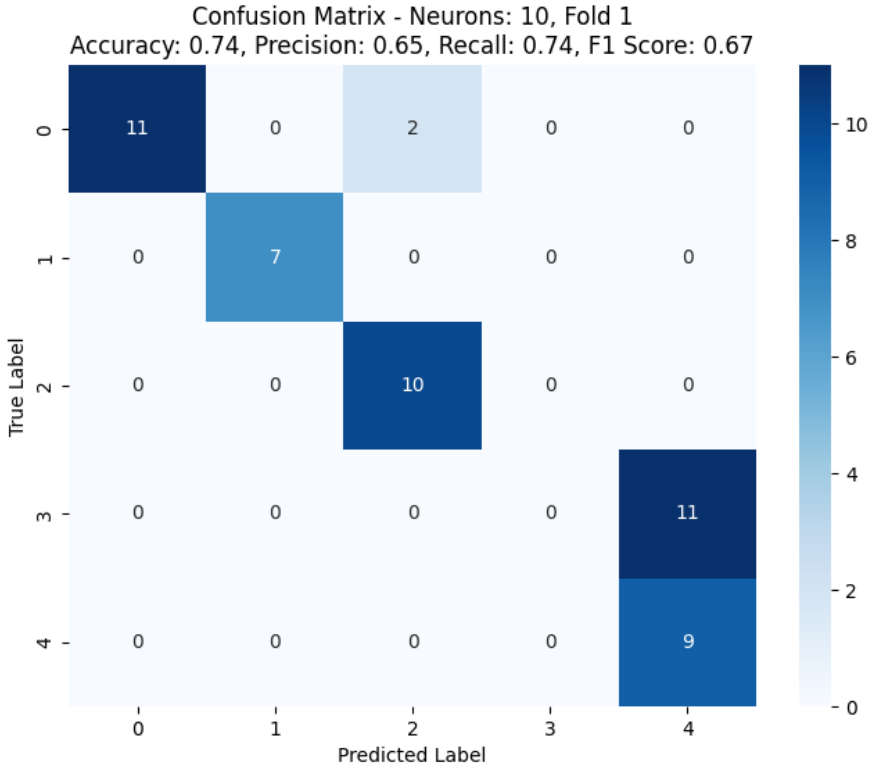

Confusion Matrix - Neurons: 5, Fold 2
Accuracy: 0.88, Precision: 0.89, Recall: 0.88, F1 Score: 0.88
True Label
Predicted Label
6 0 2 0 0
0 10 0 0 0
0 2 10 0 0
0 0 0 6 2
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

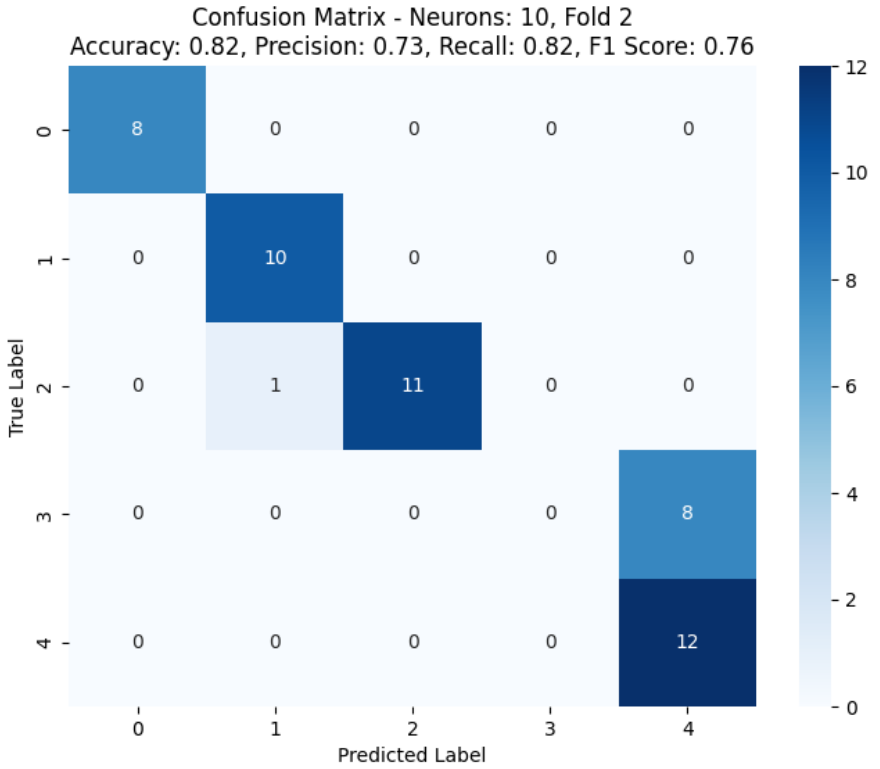

Confusion Matrix - Neurons: 5, Fold 1
Accuracy: 0.76, Precision: 0.66, Recall: 0.76, F1 Score: 0.69
True Label
Predicted Label
13 0 0 0 0
0 7 0 0 0
0 1 9 0 0
0 0 0 0 11
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10 12

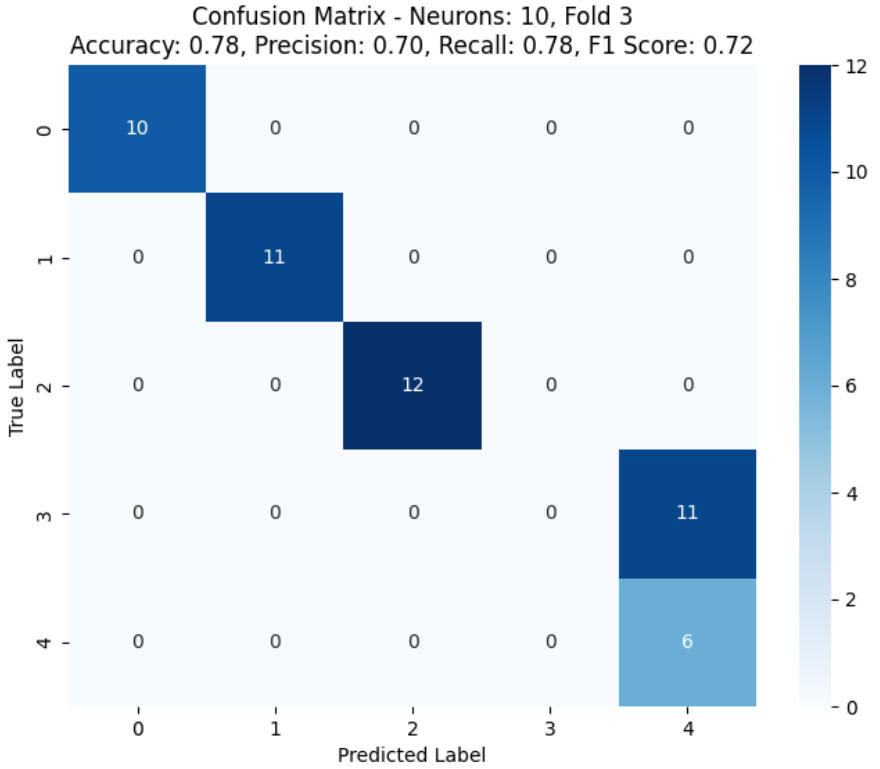

Confusion Matrix - Neurons: 5, Fold 3
Accuracy: 0.96, Precision: 0.97, Recall: 0.96, F1 Score: 0.96
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 9 2
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12

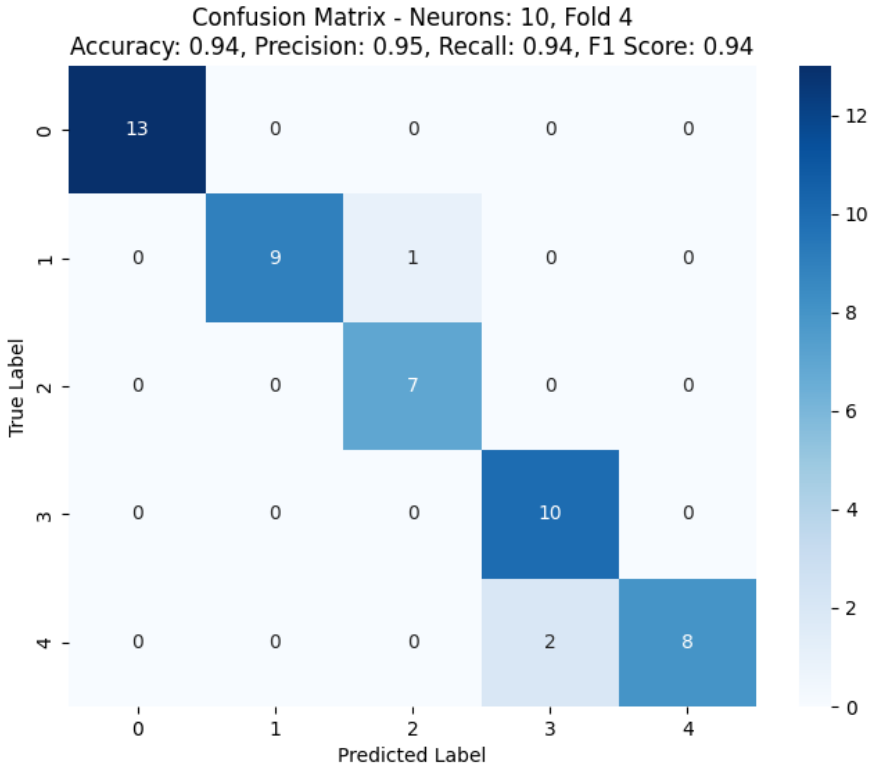

Confusion Matrix - Neurons: 5, Fold 4
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
13 0 0 0 0
0 10 0 0 0
0 2 5 0 0
0 0 0 9 1
0 0 0 1 9
0 1 2 3 4
0 2 4 6 8 10 12

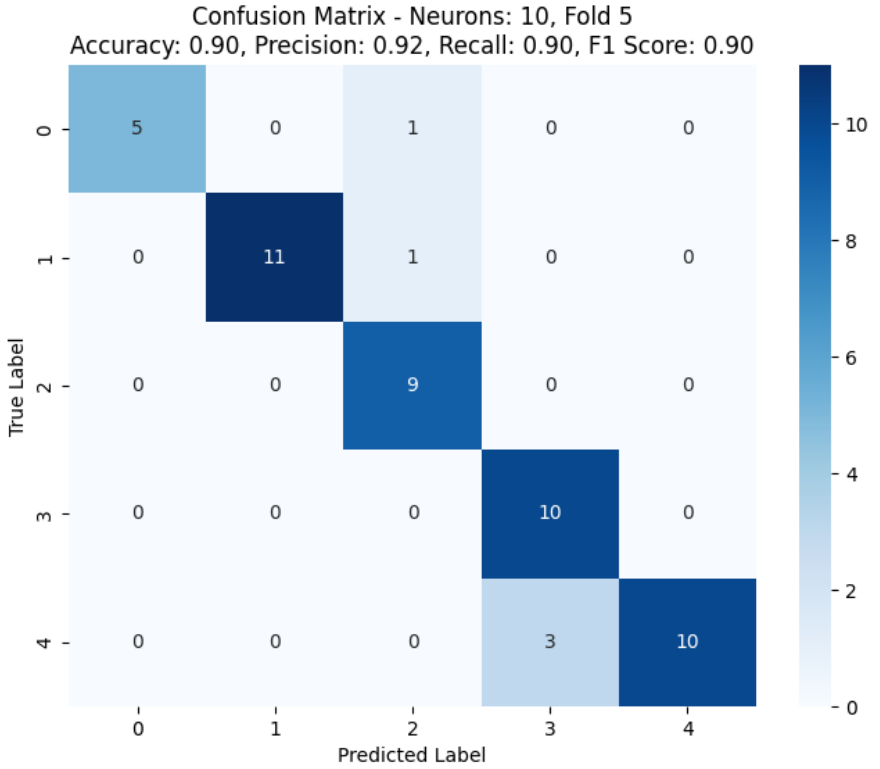

Confusion Matrix - Neurons: 5, Fold 5
Accuracy: 0.92, Precision: 0.92, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
6 0 0 0 0
0 12 0 0 0
0 0 9 0 0
0 0 0 7 3
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12

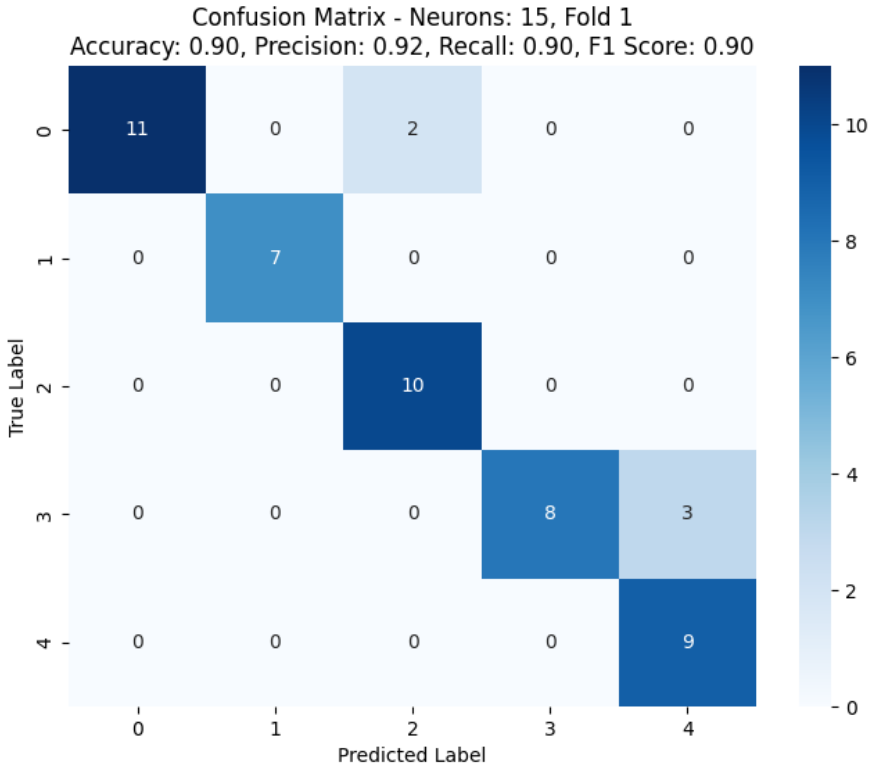

Confusion Matrix - Neurons: 10, Fold 1
Accuracy: 0.74, Precision: 0.65, Recall: 0.74, F1 Score: 0.67
True Label
Predicted Label
11 0 2 0 0
0 7 0 0 0
0 0 10 0 0
0 0 0 0 11
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10

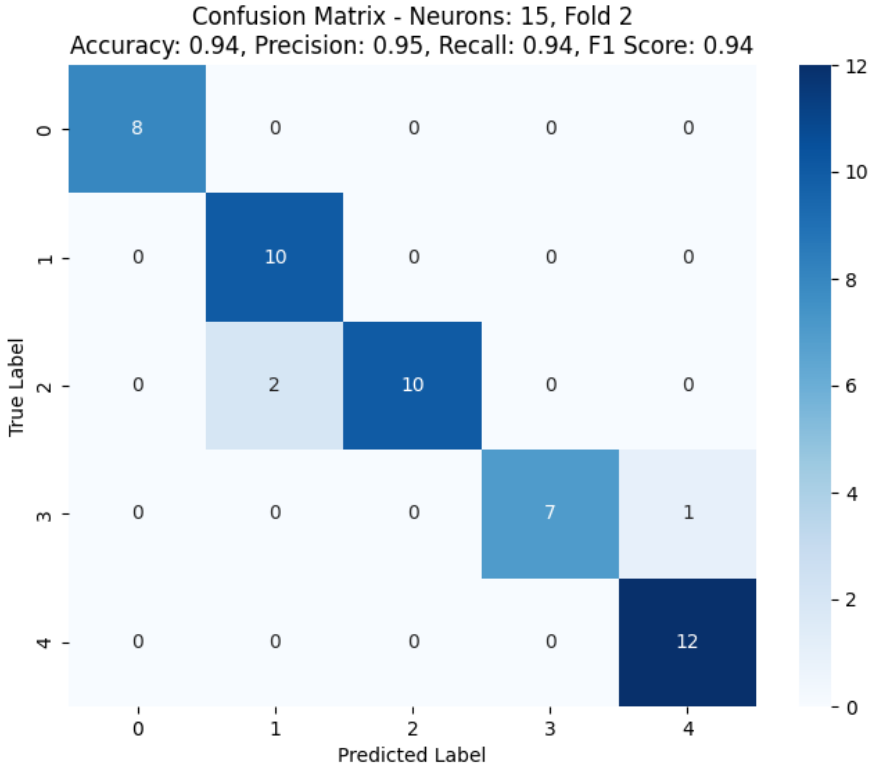

Confusion Matrix - Neurons: 10, Fold 2
Accuracy: 0.82, Precision: 0.73, Recall: 0.82, F1 Score: 0.76
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 1 11 0 0
0 0 0 0 8
0 0 0 0 12
0 1 2 3 4
12 10 8 6 4 2 0

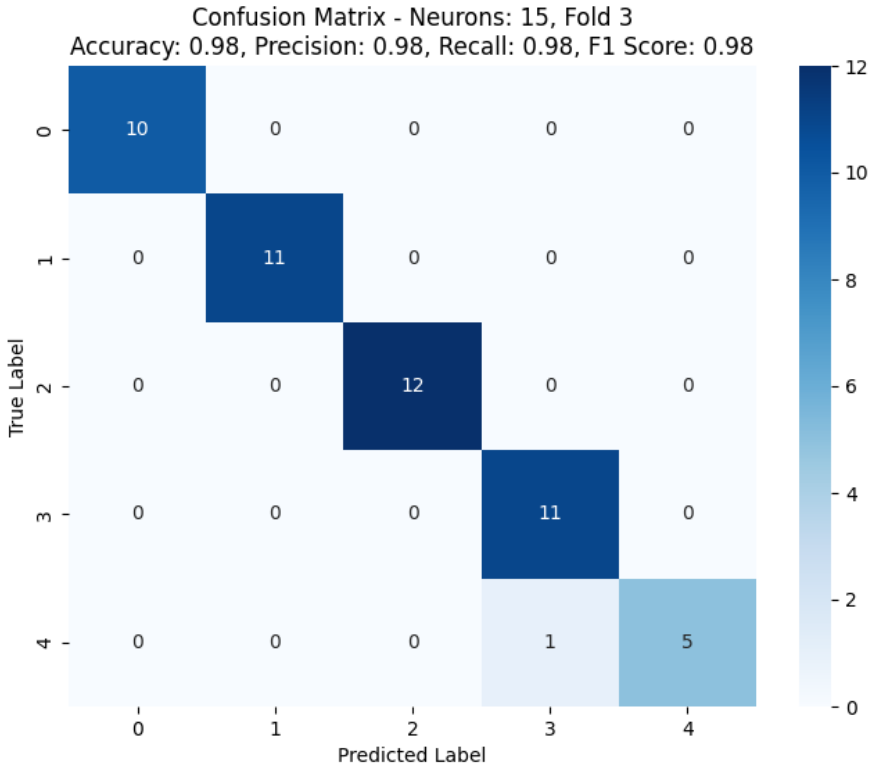

Confusion Matrix - Neurons: 10, Fold 3
Accuracy: 0.78, Precision: 0.70, Recall: 0.78, F1 Score: 0.72
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 0 11
0 0 0 0 6
0 1 2 3 4
12 10 8 6 4 2 0

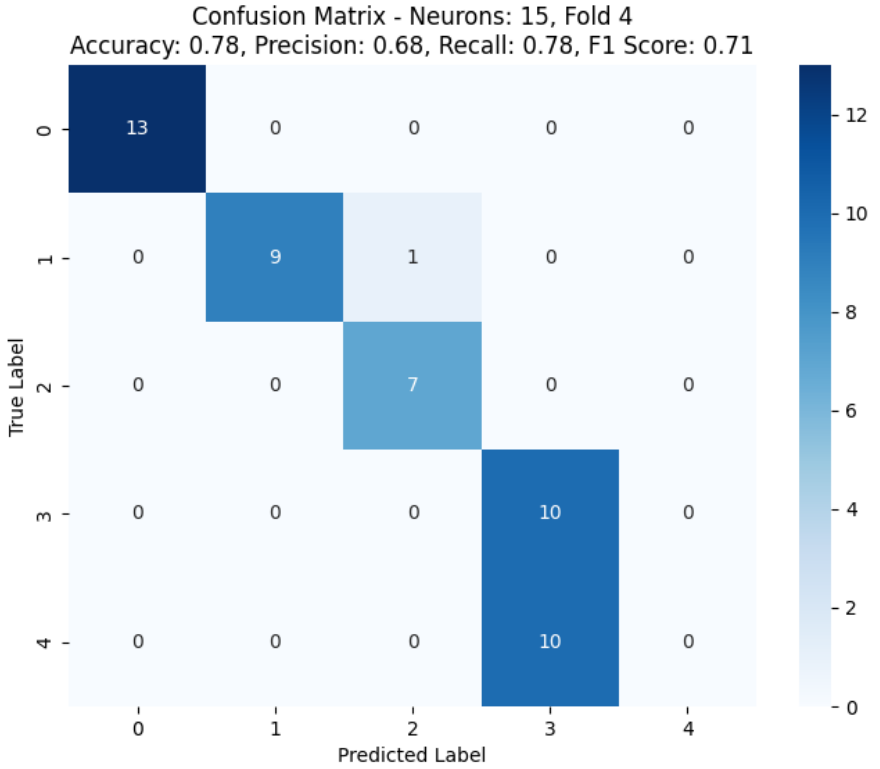

Confusion Matrix - Neurons: 10, Fold 4
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
0 0 0 10 0
0 0 0 2 8
0 1 2 3 4
0 2 4 6 8 10 12

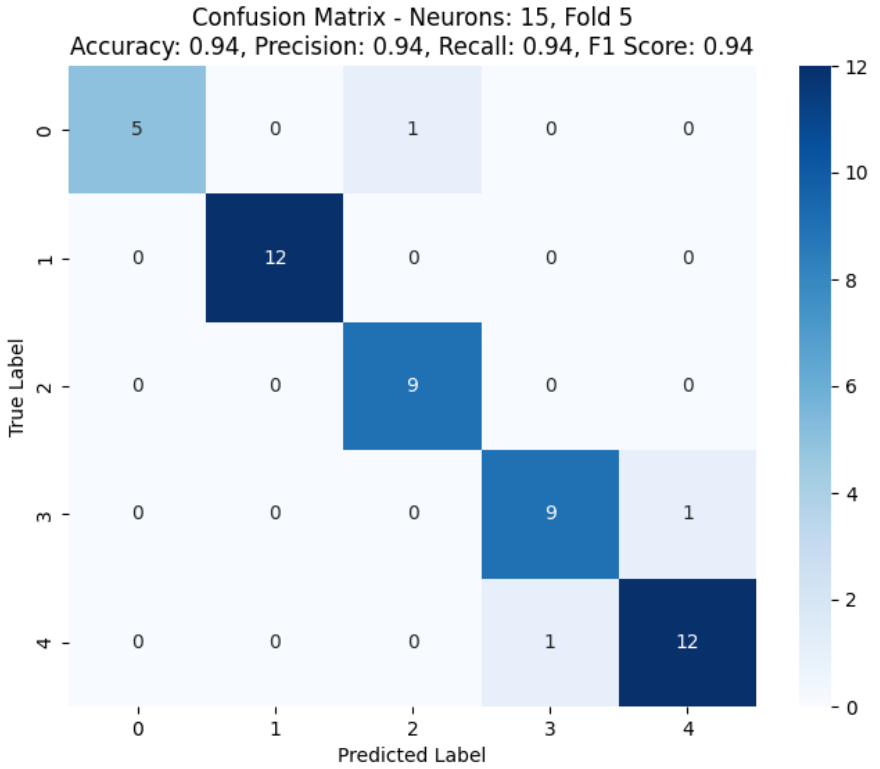

Confusion Matrix - Neurons: 10, Fold 5
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
5 0 1 0 0
0 11 1 0 0
0 0 9 0 0
0 0 0 10 0
0 0 0 3 10
0 1 2 3 4
0 2 4 6 8 10

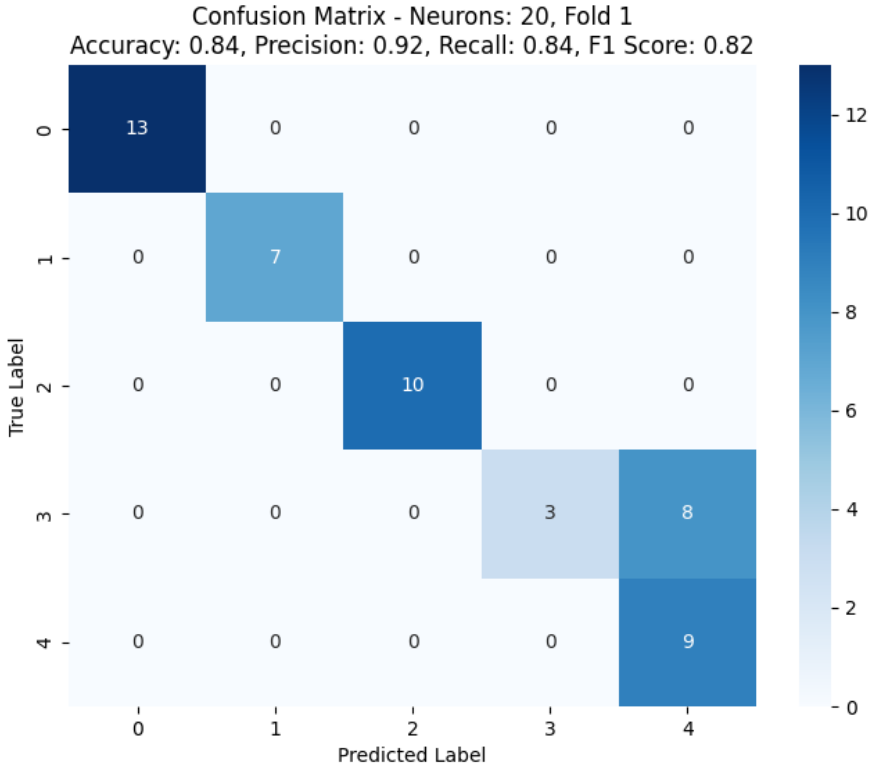

Confusion Matrix - Neurons: 15, Fold 1
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
11 0 2 0 0
0 7 0 0 0
0 0 10 0 0
0 0 0 8 3
0 0 0 0 9
0 1 2 3 4
0 2 4 6 8 10

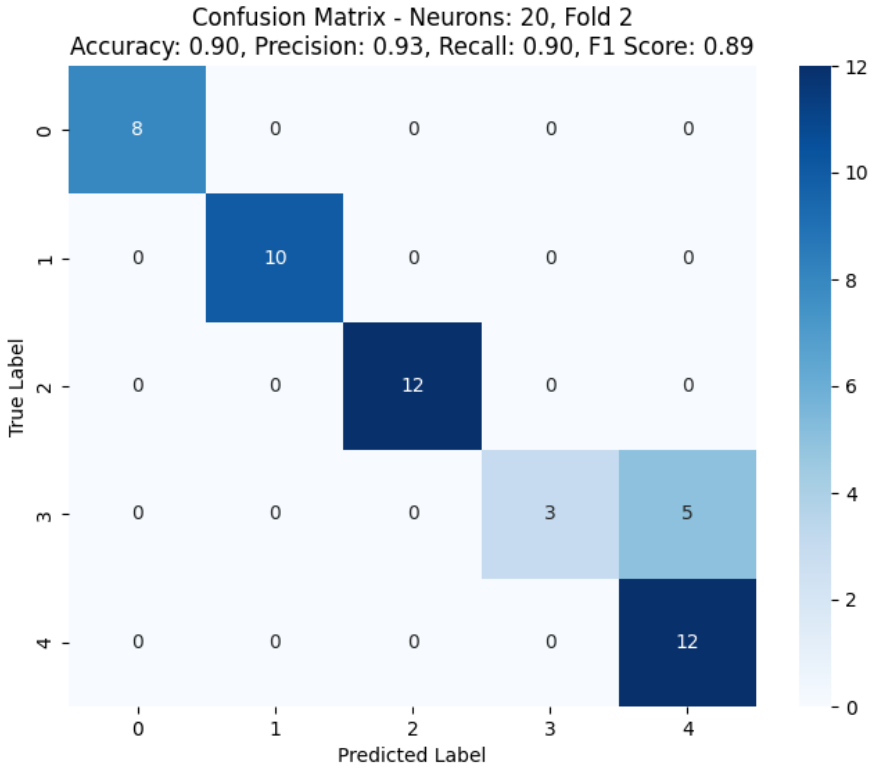

Confusion Matrix - Neurons: 15, Fold 2
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
8 0 0 0 0
0 10 0 0 0
0 2 10 0 0
0 0 0 7 1
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

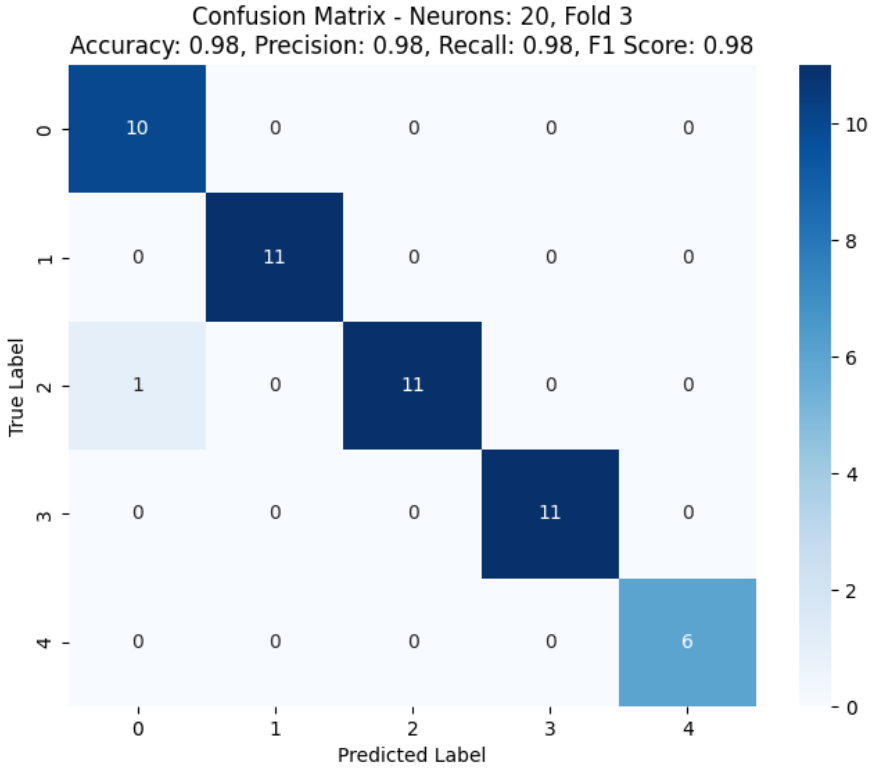

Confusion Matrix - Neurons: 15, Fold 3
Accuracy: 0.98, Precision: 0.98, Recall: 0.98, F1 Score: 0.98
True Label
Predicted Label
10 0 0 0 0
0 11 0 0 0
0 0 12 0 0
0 0 0 11 0
0 0 0 1 5
0 1 2 3 4
0 2 4 6 8 10 12

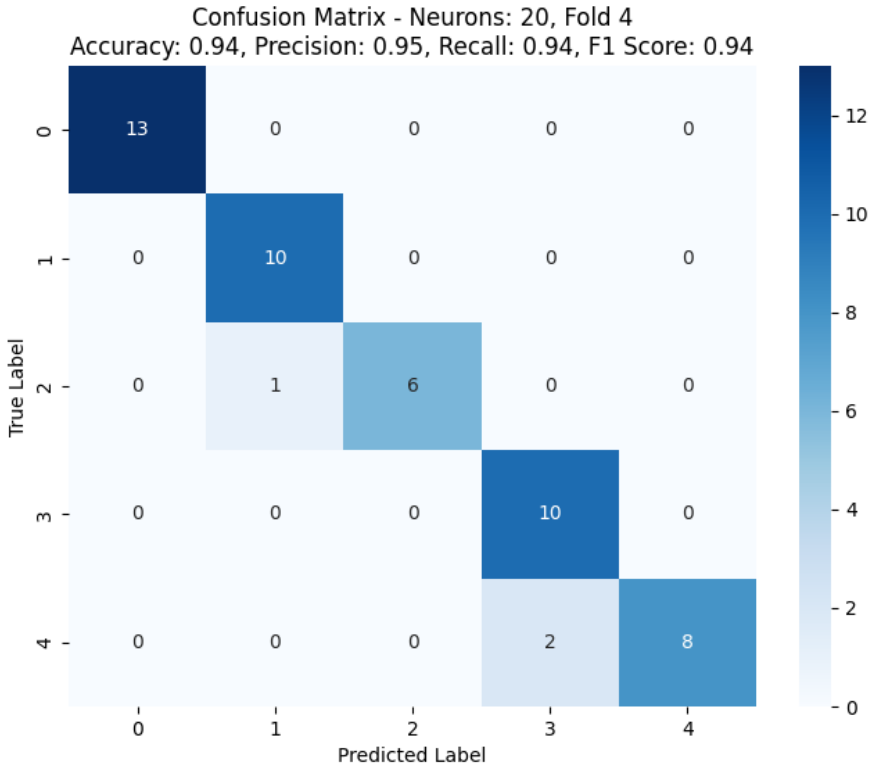

Confusion Matrix - Neurons: 15, Fold 4
Accuracy: 0.78, Precision: 0.68, Recall: 0.78, F1 Score: 0.71
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
0 0 0 10 0
0 0 0 10 0
0 1 2 3 4
0 2 4 6 8 10 12

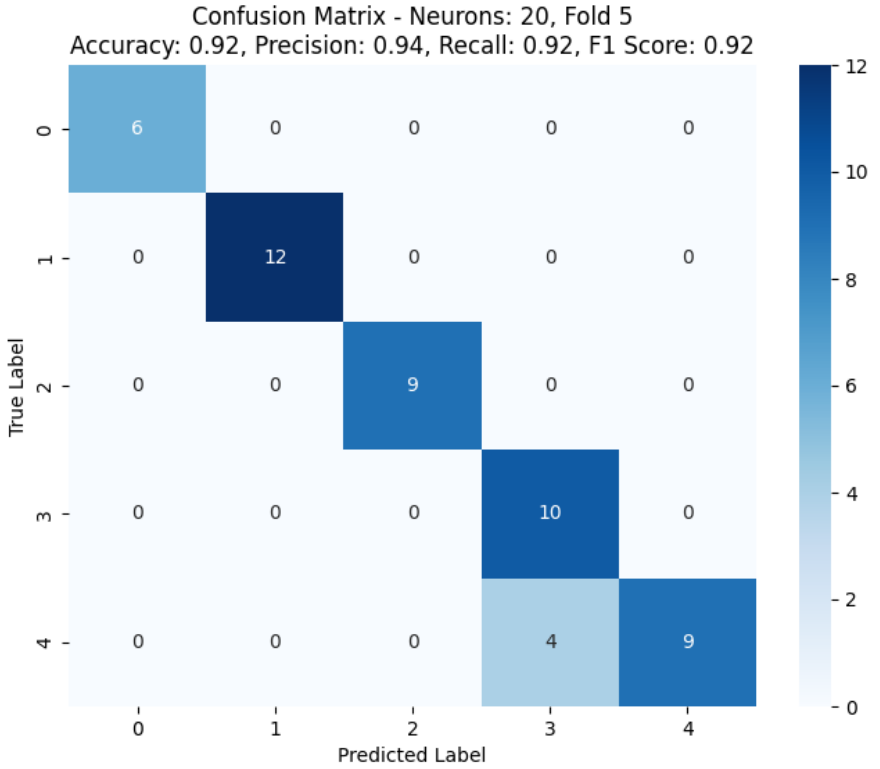

Confusion Matrix - Neurons: 15, Fold 5
Accuracy: 0.94, Precision: 0.94, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
5 0 1 0 0
0 12 0 0 0
0 0 9 0 0
0 0 0 9 1
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12


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	0.76	0.6635	0.76	0.691864	[[13 0 0 0	
3	10	1	0.74	0.647667	0.74	0.671876	[[11 0 2 0	
4	15	1	0.9	0.921667	0.9	0.8997	[[11 0 2 0	
5	20	1	0.84	0.915294	0.84	0.818901	[[13 0 0 0	
6	5	2	0.88	0.892381	0.88	0.877642	[[6 0 2 0	
7	10	2	0.82	0.725818	0.82	0.760041	[[8 0 0 0	
8	15	2	0.94	0.948205	0.94	0.939733	[[8 0 0 0	
9	20	2	0.9	0.929412	0.9	0.885893	[[8 0 0 0	
10	5	3	0.96	0.97	0.96	0.960857	[[10 0 0 0	
11	10	3	0.78	0.702353	0.78	0.722609	[[10 0 0 0	
12	15	3	0.98	0.981667	0.98	0.979526	[[10 0 0 0	
13	20	3	0.98	0.981818	0.98	0.980041	[[10 0 0 0	
14	5	4	0.92	0.926667	0.92	0.918485	[[13 0 0 0	
15	10	4	0.94	0.949167	0.94	0.939736	[[13 0 0 0	
16	15	4	0.78	0.6825	0.78	0.713474	[[13 0 0 0	
17	20	4	0.94	0.948485	0.94	0.939303	[[13 0 0 0	
18	5	5	0.92	0.923	0.92	0.918413	[[6 0 0 0	
19	10	5	0.9	0.921119	0.9	0.900656	[[5 0 1 0	
20	15	5	0.94	0.942	0.94	0.939617	[[5 0 1 0	
21	20	5	0.92	0.942857	0.92	0.919394	[[6 0 0 0	
22			average	average	average	average		
23	5		0.888	0.8751	0.888	0.8735		
24	10		0.836	0.7892	0.836	0.799		
25	15		0.908	0.8952	0.908	0.8944		
26	20		0.916	0.9436	0.916	0.9087		

Đánh giá hệ thống sử dụng phương pháp ANN-HOG với 5-fold cross-validation:
Hiệu quả tổng quan:

Hệ thống ANN-HOG được đánh giá dựa trên sự thay đổi số lượng neuron ẩn từ 5 đến 20. Nhìn chung, khi số lượng neuron ẩn tăng, hiệu suất hệ thống cũng tăng lên, được thể hiện rõ qua các chỉ số như accuracy, precision, recall, và f1 score.
Sự ảnh hưởng của số lượng neuron ẩn đến hệ thống:

5 Neuron ẩn: Hệ thống đạt accuracy trung bình là 0.888 với precision là 0.8751. Mặc dù đạt hiệu quả khá cao, việc sử dụng ít neuron ẩn có thể dẫn đến việc mô hình chưa học được đầy đủ các đặc trưng phức tạp của dữ liệu.

10 Neuron ẩn: Hiệu suất giảm nhẹ với accuracy trung bình là 0.836. Số neuron này có vẻ không đủ để hệ thống phân tích hết các đặc trưng của dữ liệu, dẫn đến kết quả kém hơn.
15 Neuron ẩn: Khi số neuron tăng lên 15, hệ thống cho thấy hiệu suất rất tốt, với accuracy đạt 0.908 và f1 score là 0.8944. Điều này cho thấy hệ thống đang bắt đầu tối ưu hóa với số lượng neuron lớn hơn, giúp mô hình học được nhiều đặc trưng hơn và cải thiện kết quả.
20 Neuron ẩn: Hệ thống tiếp tục cải thiện với accuracy đạt 0.916 và precision tăng lên mức cao nhất (0.9436). Tuy nhiên, sự khác biệt về hiệu suất giữa 15 và 20 neuron là không quá lớn, cho thấy mô hình đã bắt đầu đạt đến ngưỡng tốt nhất khi số lượng neuron tăng.
Tính cân bằng giữa các chỉ số:

Với 5 và 10 neuron ẩn, precision thấp hơn một chút so với recall, cho thấy hệ thống có thể đang bỏ lỡ một số mẫu dương tính. Tuy nhiên, khi tăng số lượng neuron lên 15 và 20, precision vượt trội, cho thấy hệ thống có khả năng nhận diện chính xác các mẫu.
f1 score tăng đều khi số neuron tăng, đặc biệt cao nhất với 20 neuron ẩn (0.9087), cho thấy mô hình đã đạt sự cân bằng giữa precision và recall.
Kết luận:
Số lượng neuron ẩn có tác động rõ rệt đến hiệu suất của hệ thống ANN-HOG. Khi tăng số lượng neuron từ 5 lên 20, các chỉ số như accuracy, precision, recall, và f1 score đều cải thiện đáng kể.
Hệ thống đạt hiệu suất tối ưu với 15 đến 20 neuron ẩn, khi các chỉ số như accuracy đạt trên 0.9, cho thấy mô hình đã học tốt và phân loại chính xác hơn với số lượng neuron lớn hơn. Tuy nhiên, sau mức 15 neuron, sự cải thiện không quá lớn, có thể xem xét mức tối ưu là 15 neuron để đạt hiệu quả và tối ưu hóa tài nguyên tính toán.

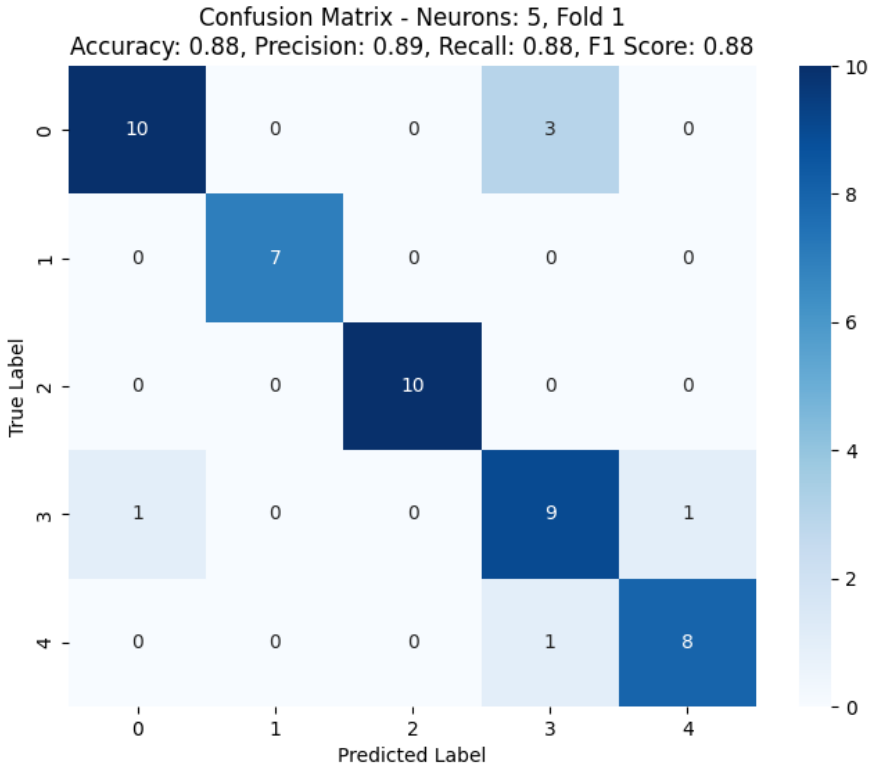
ANN-HU:

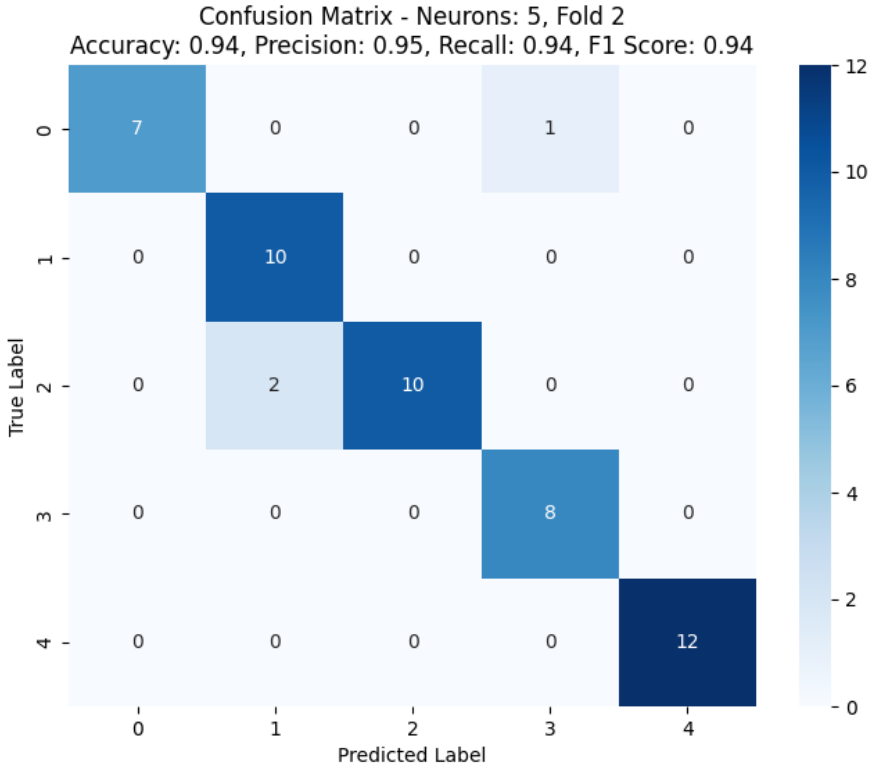
```
# Bước 1: Tải dữ liệu từ file CSV
data = pd.read_csv(r'E:/Downloads/DATA/Hu/hutest_hsv/HUnhom11Std.csv')

# 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)
```

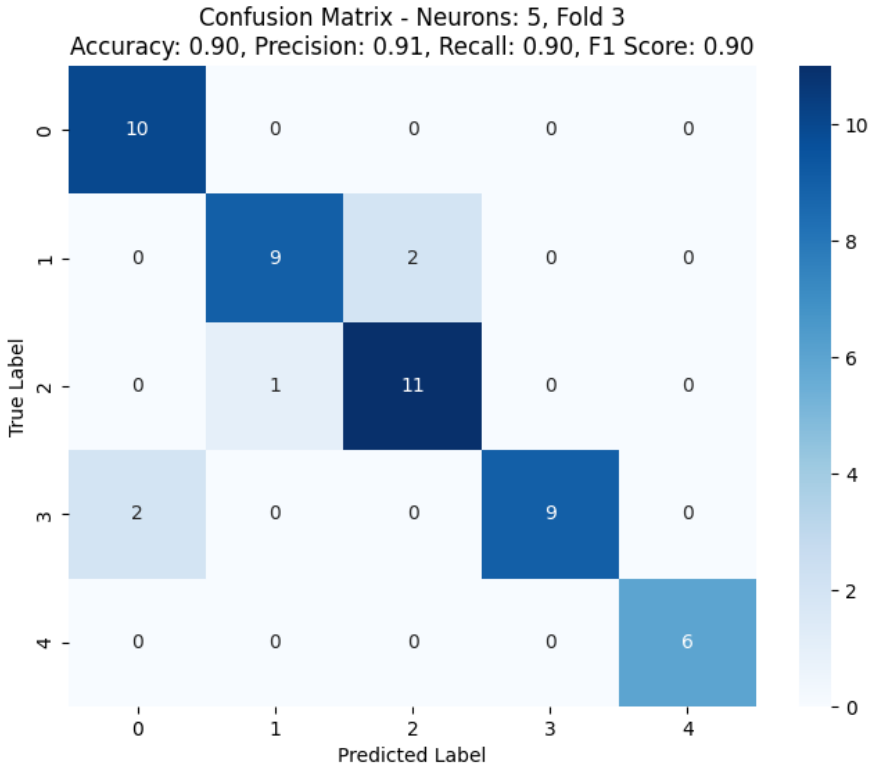
```
# Danh sách số neuron để thử nghiệm
neurons_list = [5, 10, 15, 20]

# 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,
```

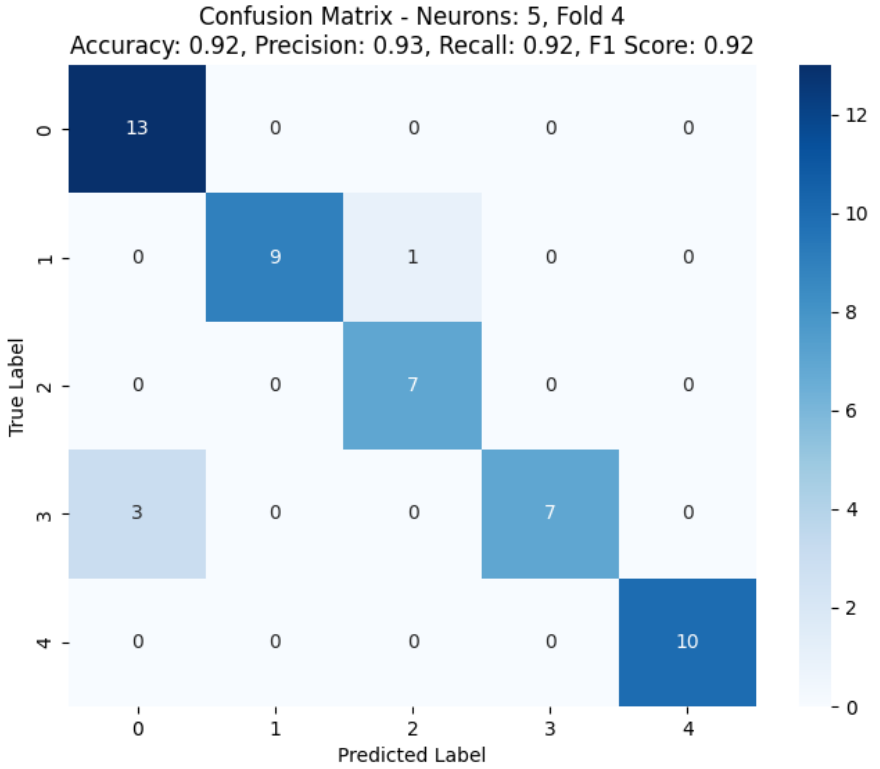
```
            'f1': f1,
            'confusion_matrix': conf_matrix
        })

        # 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)
```

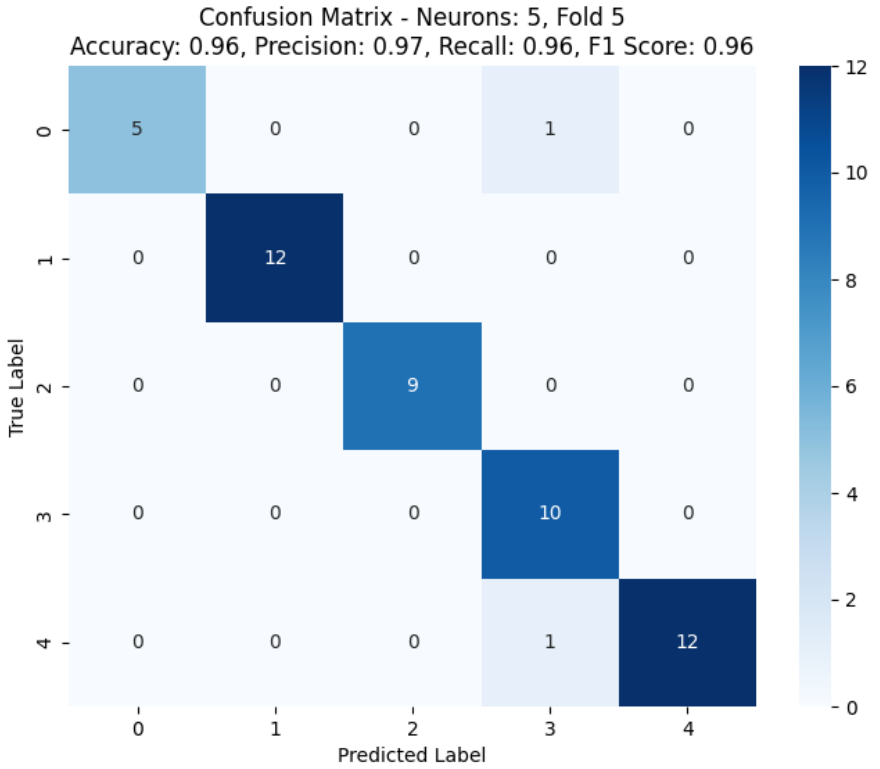
Kết quả:
Ma trận nhầm lẫn

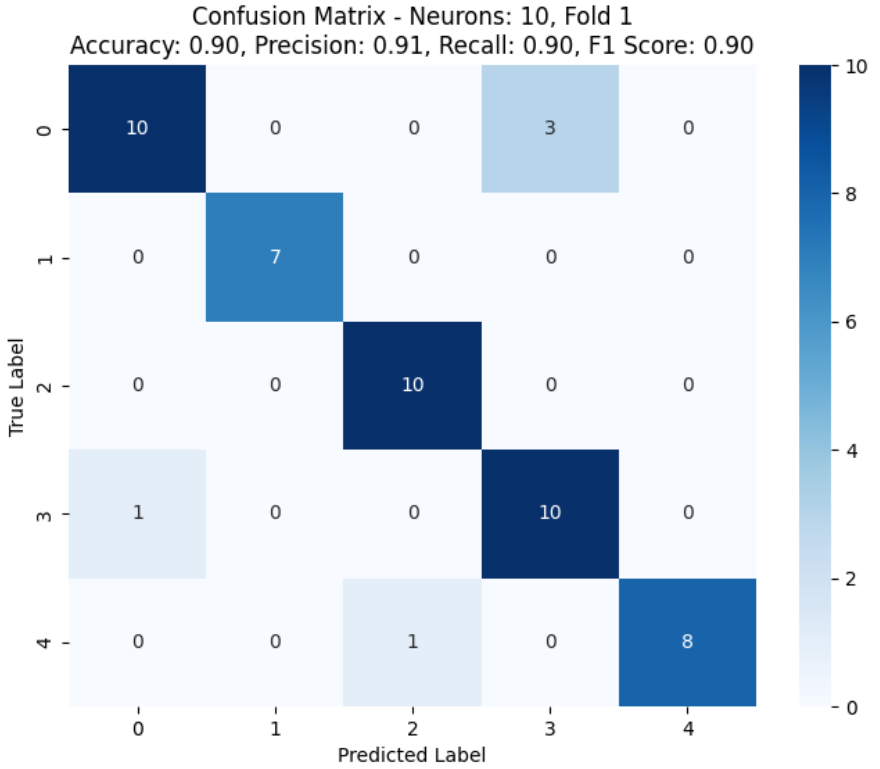

Confusion Matrix - Neurons: 5, Fold 1
Accuracy: 0.88, Precision: 0.89, Recall: 0.88, F1 Score: 0.88
True Label
Predicted Label
10 0 0 3 0
0 7 0 0 0
0 0 10 0 0
1 0 0 9 1
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10

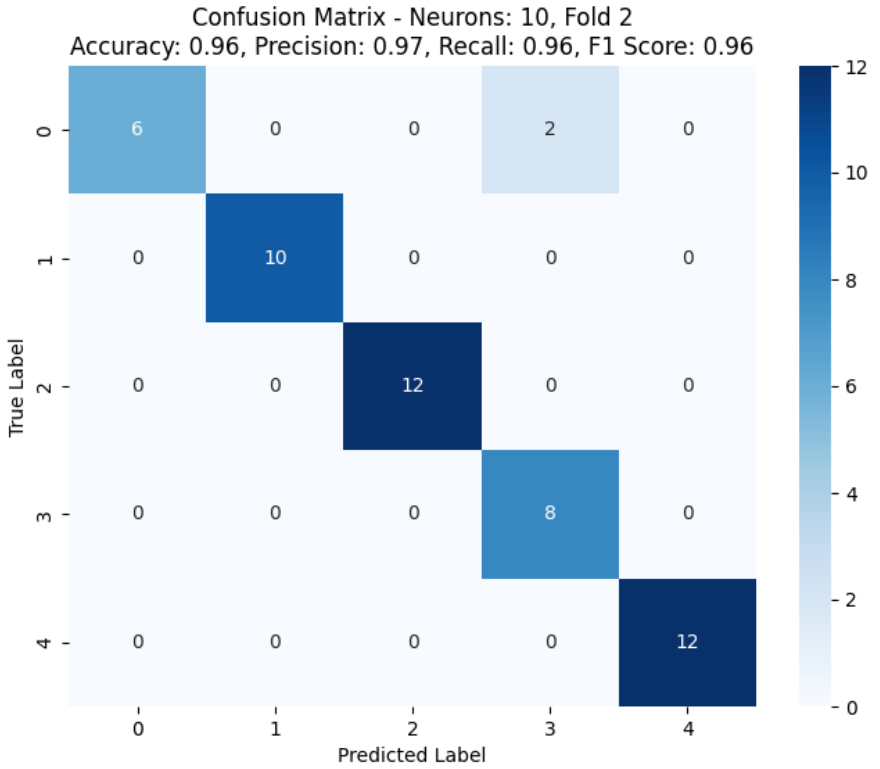

Confusion Matrix - Neurons: 5, Fold 2
Accuracy: 0.94, Precision: 0.95, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
7 0 0 1 0
0 10 0 0 0
0 2 10 0 0
0 0 0 8 0
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

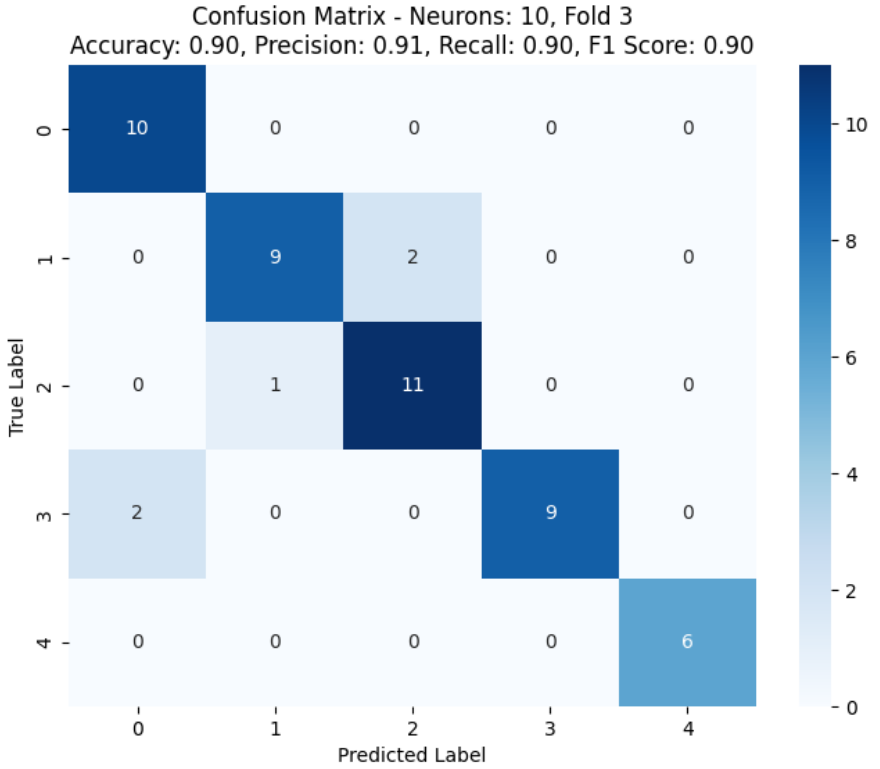

Confusion Matrix - Neurons: 5, Fold 3
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
10 0 0 0 0
0 9 2 0 0
0 1 11 0 0
2 0 0 9 0
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10

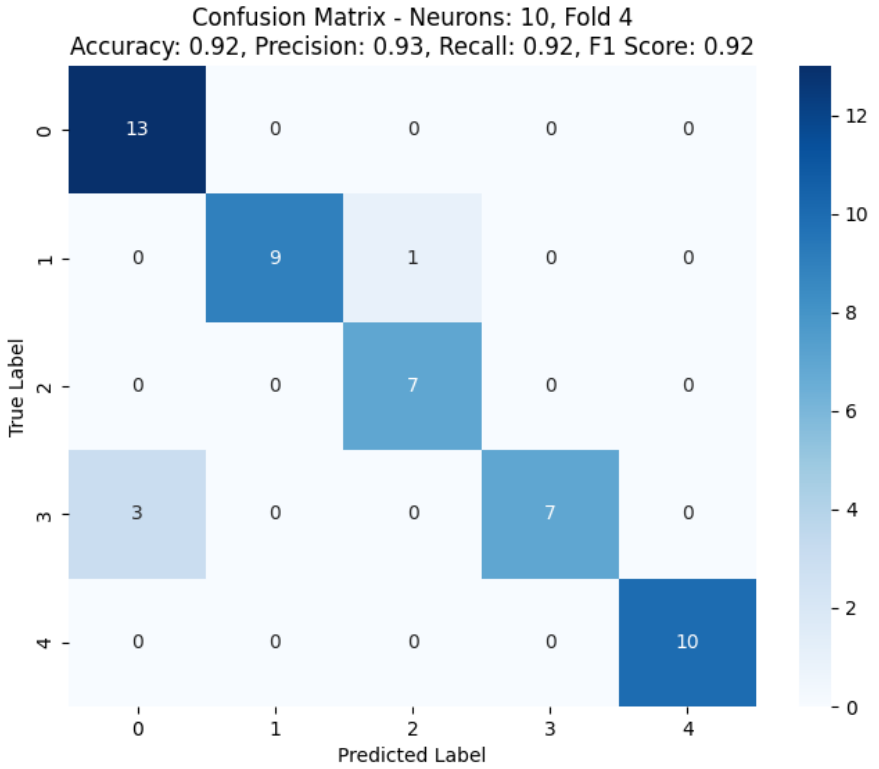

Confusion Matrix - Neurons: 5, Fold 4
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
3 0 0 7 0
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

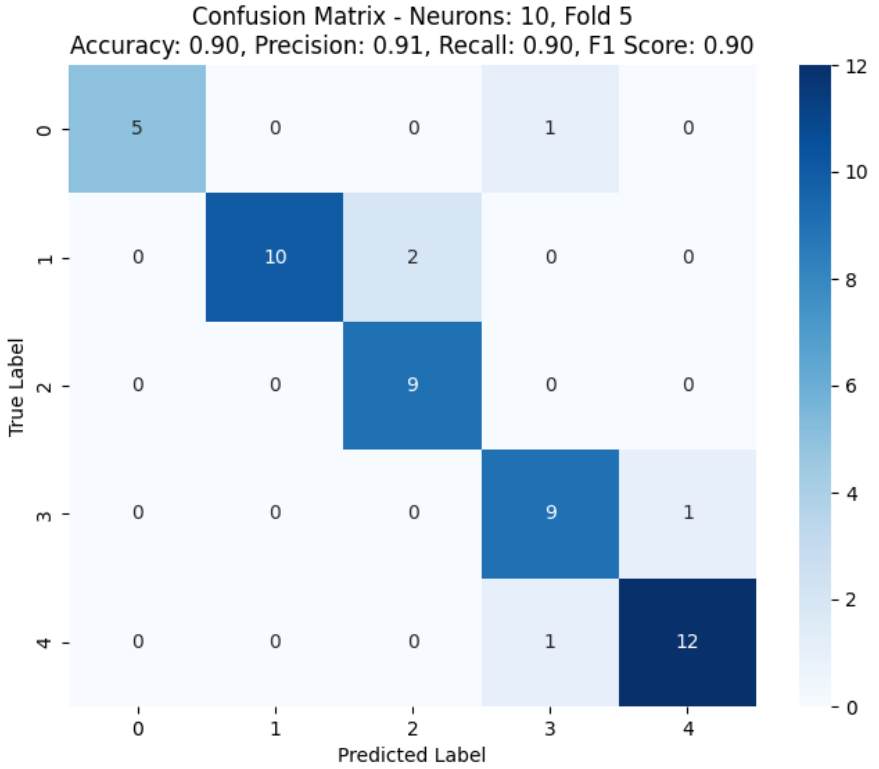

Confusion Matrix - Neurons: 5, Fold 5
Accuracy: 0.96, Precision: 0.97, Recall: 0.96, F1 Score: 0.96
True Label
Predicted Label
5 0 0 1 0
0 12 0 0 0
0 0 9 0 0
0 0 0 10 0
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12

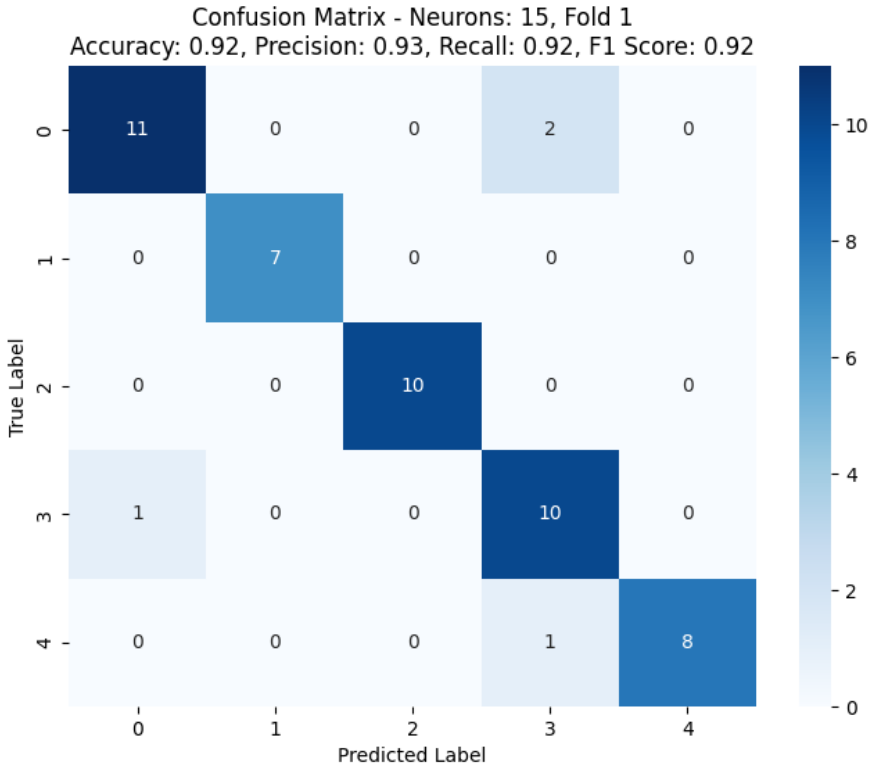

Confusion Matrix - Neurons: 10, Fold 1
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
10 0 0 3 0
0 7 0 0 0
0 0 10 0 0
1 0 0 10 0
0 0 1 0 8
0 1 2 3 4
0 2 4 6 8 10

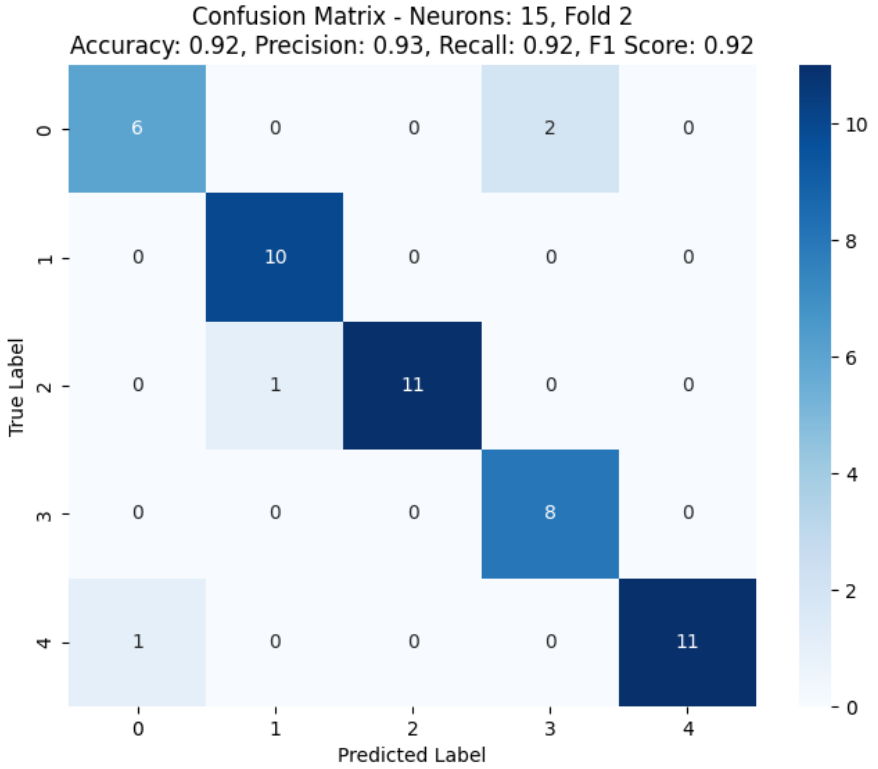

Confusion Matrix - Neurons: 10, Fold 2
Accuracy: 0.96, Precision: 0.97, Recall: 0.96, F1 Score: 0.96
True Label
Predicted Label
6 0 0 2 0
0 10 0 0 0
0 0 12 0 0
0 0 0 8 0
0 0 0 0 12
0 1 2 3 4
0 2 4 6 8 10 12

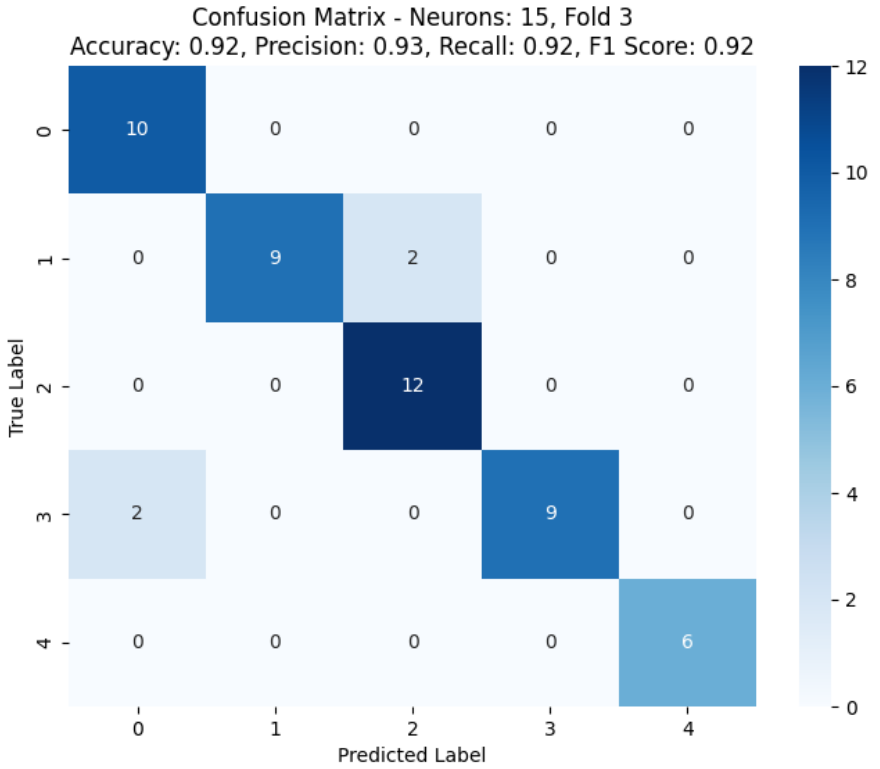

Confusion Matrix - Neurons: 10, Fold 3
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
10 0 0 0 0
0 9 2 0 0
0 1 11 0 0
2 0 0 9 0
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10

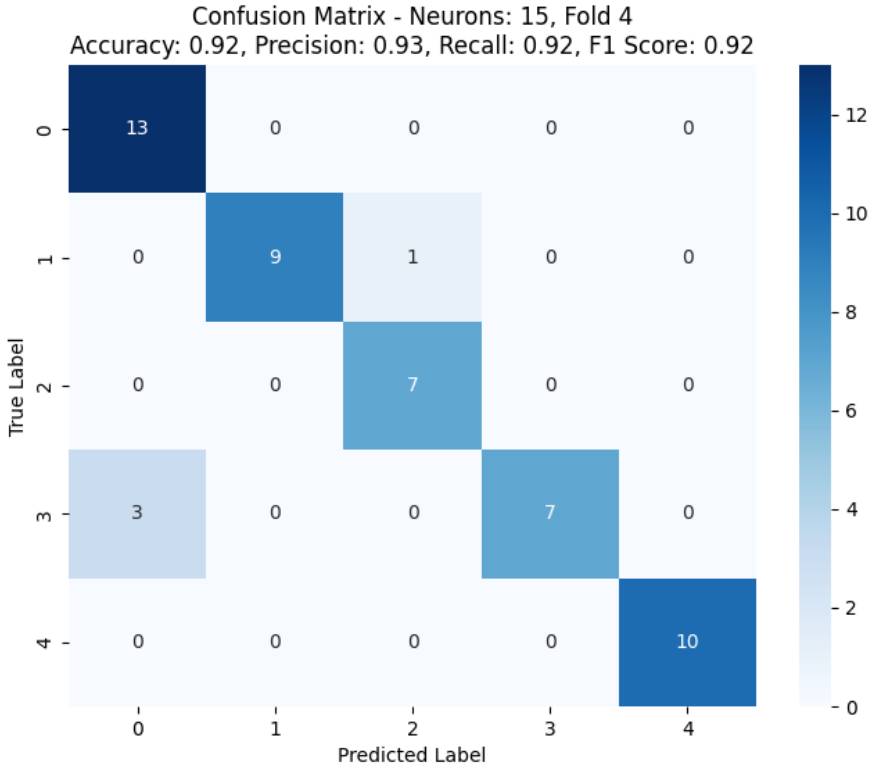

Confusion Matrix - Neurons: 10, Fold 4
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
3 0 0 7 0
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

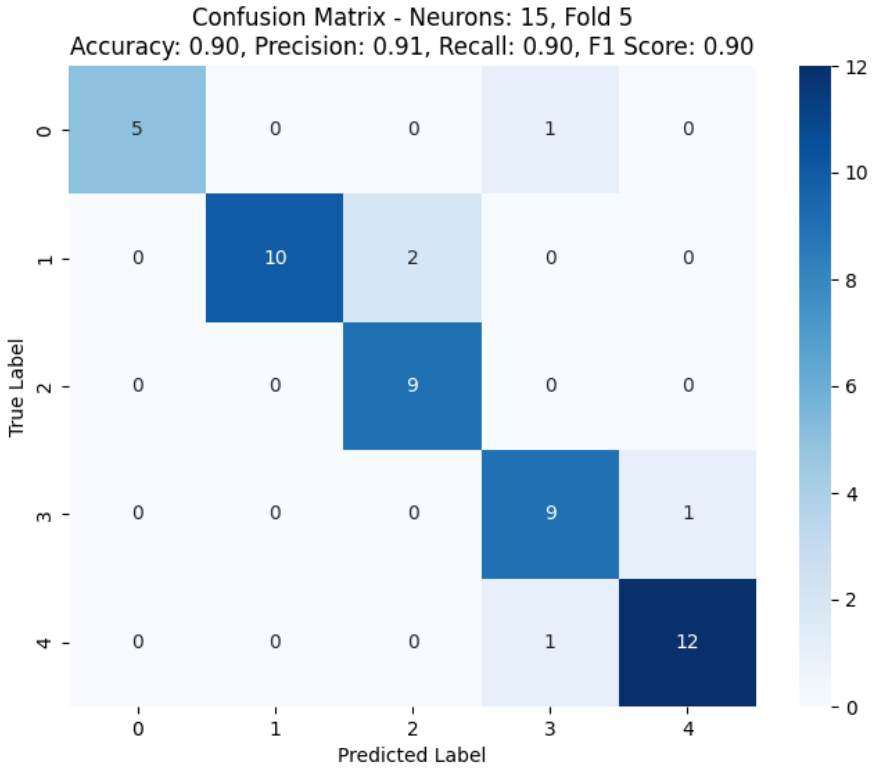

Confusion Matrix - Neurons: 10, Fold 5
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
5 0 0 1 0
0 10 2 0 0
0 0 9 0 0
0 0 0 9 1
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12

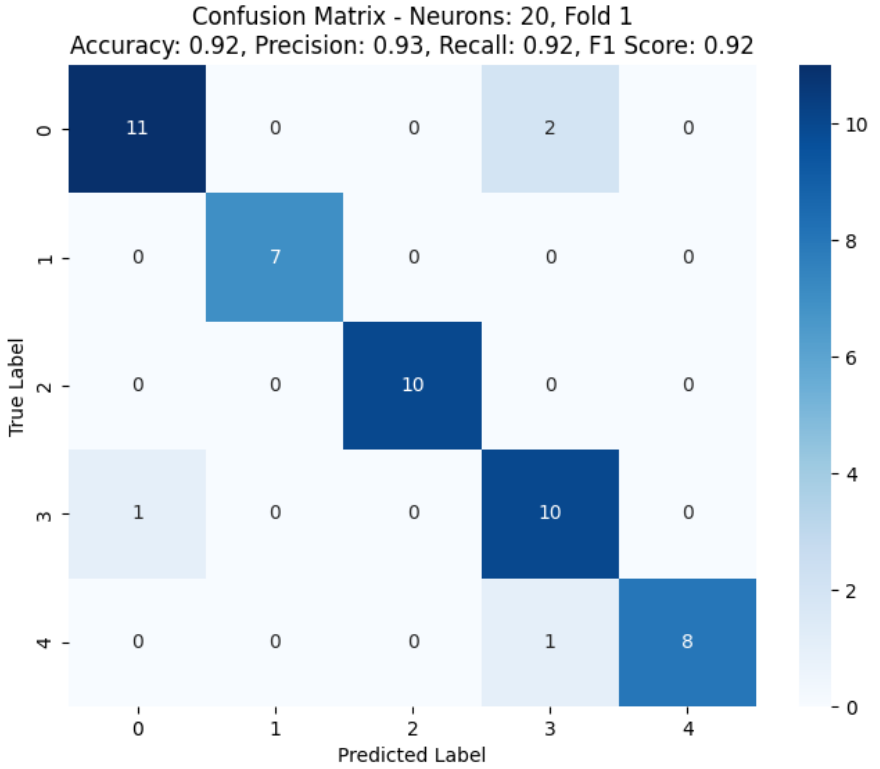

Confusion Matrix - Neurons: 15, Fold 1
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
11 0 0 2 0
0 7 0 0 0
0 0 10 0 0
1 0 0 10 0
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10

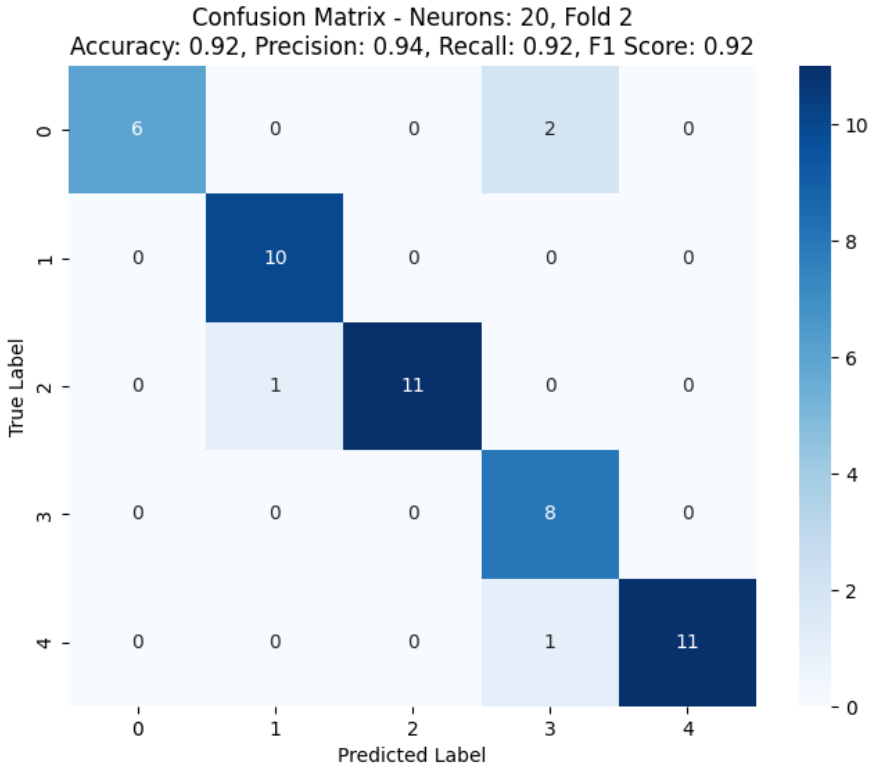

Confusion Matrix - Neurons: 15, Fold 2
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
6 0 0 2 0
0 10 0 0 0
0 1 11 0 0
0 0 0 8 0
1 0 0 0 11
0 1 2 3 4
0 2 4 6 8 10

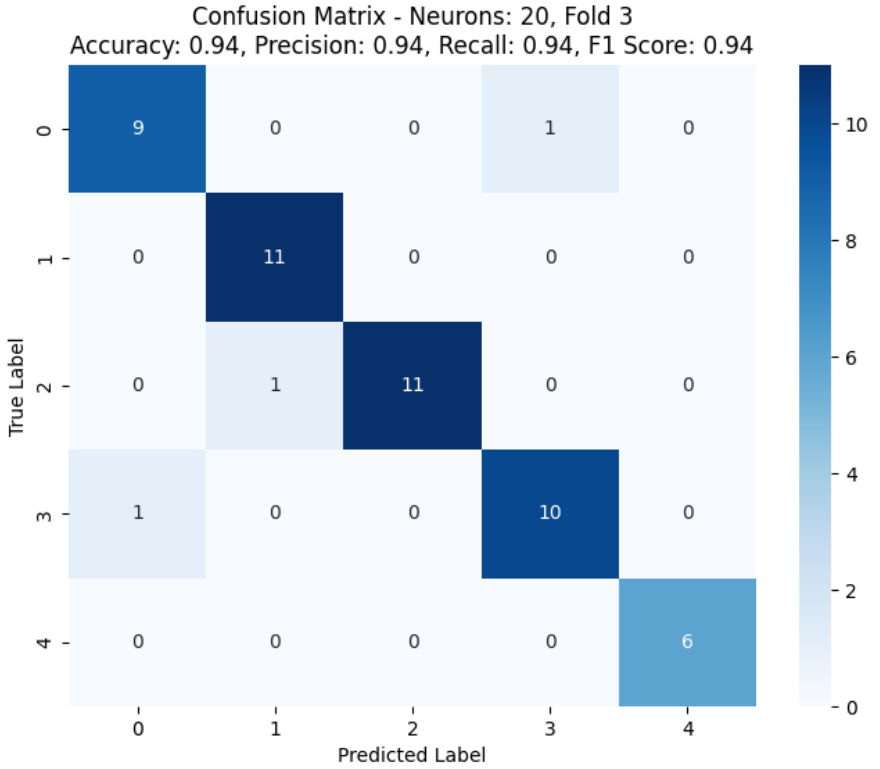

Confusion Matrix - Neurons: 15, Fold 3
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
10 0 0 0 0
0 9 2 0 0
0 0 12 0 0
2 0 0 9 0
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10 12

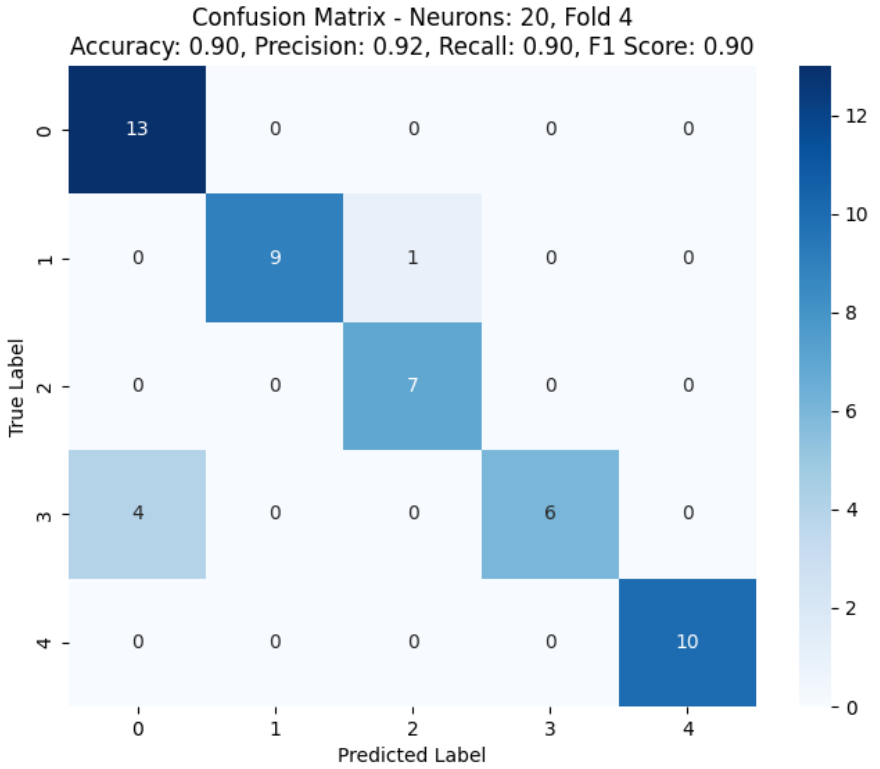

Confusion Matrix - Neurons: 15, Fold 4
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
3 0 0 7 0
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12

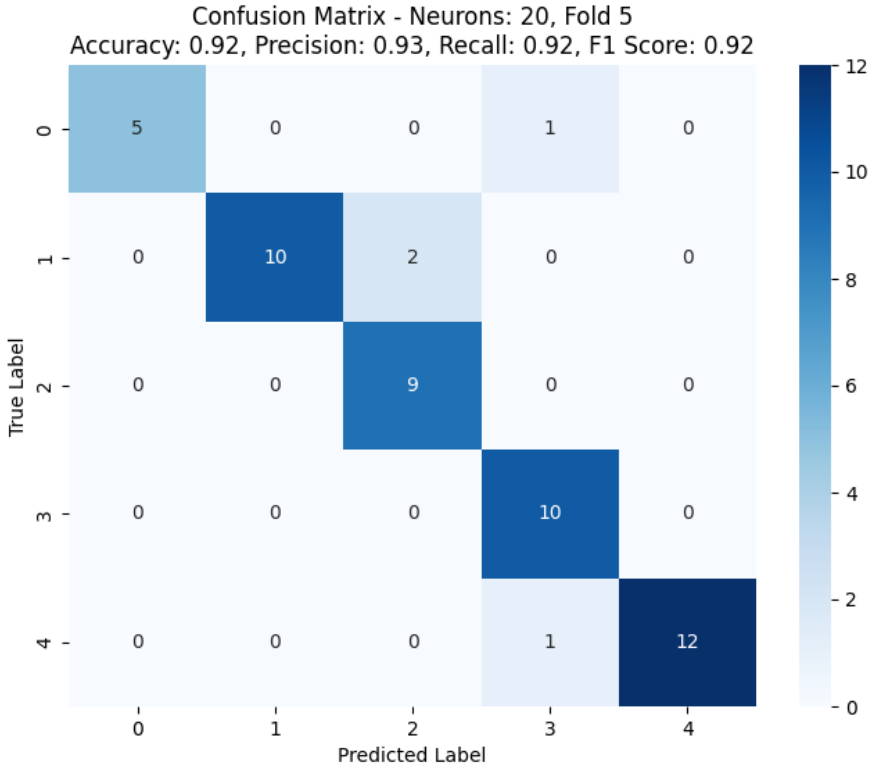

Confusion Matrix - Neurons: 15, Fold 5
Accuracy: 0.90, Precision: 0.91, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
5 0 0 1 0
0 10 2 0 0
0 0 9 0 0
0 0 0 9 1
0 0 0 1 12
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 1
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
11 0 0 2 0
0 7 0 0 0
0 0 10 0 0
1 0 0 10 0
0 0 0 1 8
0 1 2 3 4
0 2 4 6 8 10


Confusion Matrix - Neurons: 20, Fold 2
Accuracy: 0.92, Precision: 0.94, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
6 0 0 2 0
0 10 0 0 0
0 1 11 0 0
0 0 0 8 0
0 0 0 1 11
0 1 2 3 4
0 2 4 6 8 10


Confusion Matrix - Neurons: 20, Fold 3
Accuracy: 0.94, Precision: 0.94, Recall: 0.94, F1 Score: 0.94
True Label
Predicted Label
9 0 0 1 0
0 11 0 0 0
0 1 11 0 0
1 0 0 10 0
0 0 0 0 6
0 1 2 3 4
0 2 4 6 8 10


Confusion Matrix - Neurons: 20, Fold 4
Accuracy: 0.90, Precision: 0.92, Recall: 0.90, F1 Score: 0.90
True Label
Predicted Label
13 0 0 0 0
0 9 1 0 0
0 0 7 0 0
4 0 0 6 0
0 0 0 0 10
0 1 2 3 4
0 2 4 6 8 10 12


Confusion Matrix - Neurons: 20, Fold 5
Accuracy: 0.92, Precision: 0.93, Recall: 0.92, F1 Score: 0.92
True Label
Predicted Label
5 0 0 1 0
0 10 2 0 0
0 0 9 0 0
0 0 0 10 0
0 0 0 1 12
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	0.88	0.888671	0.88	0.881667	[[10 0 0 3	
3	10	1	0.9	0.907413	0.9	0.899888	[[10 0 0 3	
4	15	1	0.92	0.927564	0.92	0.921545	[[11 0 0 2	
5	20	1	0.92	0.927564	0.92	0.921545	[[11 0 0 2	
6	5	2	0.94	0.948889	0.94	0.939922	[[7 0 0 1	
7	10	2	0.96	0.968	0.96	0.959365	[[6 0 0 2	
8	15	2	0.92	0.926961	0.92	0.919829	[[6 0 0 2	
9	20	2	0.92	0.938182	0.92	0.921486	[[6 0 0 2	
10	5	3	0.9	0.907744	0.9	0.89959	[[10 0 0 0	
11	10	3	0.9	0.907744	0.9	0.89959	[[10 0 0 0	
12	15	3	0.92	0.932381	0.92	0.919357	[[10 0 0 0	
13	20	3	0.94	0.941667	0.94	0.94	[[9 0 0 1	
14	5	4	0.92	0.93375	0.92	0.91795	[[13 0 0 0	
15	10	4	0.92	0.93375	0.92	0.91795	[[13 0 0 0	
16	15	4	0.92	0.93375	0.92	0.91795	[[13 0 0 0	
17	20	4	0.9	0.921324	0.9	0.895474	[[13 0 0 0	
18	5	5	0.96	0.966667	0.96	0.960509	[[5 0 0 1	
19	10	5	0.9	0.910909	0.9	0.900701	[[5 0 0 1	
20	15	5	0.9	0.910909	0.9	0.900701	[[5 0 0 1	
21	20	5	0.92	0.933939	0.92	0.920691	[[5 0 0 1	
22			average	average	average	average		
23	5		0.916	0.9291	0.912	0.9204		
24	10		0.92	0.9256	0.912	0.9136		
25	15		0.92	0.9263	0.916	0.9158		
26	20		0.92	0.9325	0.924	0.9198		

Đánh giá hệ thống sử dụng ANN-Hu's Moments với 5-fold cross-validation:

Hiệu quả tổng quan:

Hệ thống được đánh giá dựa trên sự thay đổi số lượng neuron ẩn từ 5 đến 20. Nhìn chung, với cả 4 cấu hình số lượng neuron, hệ thống đều đạt accuracy cao (dao động từ 0.916 đến 0.92) và có sự ổn định giữa các chỉ số precision, recall, và f1 score.

Sự ảnh hưởng của số lượng neuron ẩn đến hệ thống:

5 Neuron ẩn: Hệ thống đạt accuracy trung bình là 0.916 và f1 score là 0.9204. Hệ thống hoạt động khá ổn định với ít neuron, nhưng có thể còn chưa tối ưu trong việc phân loại một số mẫu phức tạp hơn.

10 Neuron ẩn: Hiệu suất được cải thiện nhẹ với accuracy trung bình đạt 0.92. Mặc dù precision giảm nhẹ so với số neuron ít hơn (0.9256), nhưng f1 score vẫn khá ổn định ở mức 0.9136. Số lượng neuron này đã giúp hệ thống bắt đầu phân loại dữ liệu tốt hơn, nhưng vẫn chưa phải là tối ưu.

15 Neuron ẩn: Với số lượng neuron tăng lên 15, hệ thống có hiệu suất tốt hơn với accuracy trung bình vẫn đạt 0.92 và f1 score đạt 0.9158. Điều này cho thấy mô hình bắt đầu học được các đặc trưng phức tạp hơn trong dữ liệu.

20 Neuron ẩn: Tăng số neuron lên 20 giúp hệ thống đạt precision cao nhất (0.9325), cùng với accuracy duy trì ở mức cao (0.92). Tuy nhiên, sự cải thiện giữa 15 và 20 neuron là không quá rõ ràng, với f1 score chỉ tăng nhẹ (0.9198).

Tính cân bằng giữa các chỉ số:

Khi số lượng neuron ẩn tăng từ 5 đến 20, các chỉ số precision, recall, và f1 score đều được cải thiện. Điều này cho thấy hệ thống đang ngày càng phân loại chính xác hơn các mẫu dữ liệu và giảm bớt lỗi.

Precision là chỉ số có sự cải thiện rõ rệt nhất khi số neuron tăng, điều này cho thấy khả năng nhận diện chính xác các mẫu dương tính được nâng cao khi hệ thống có nhiều neuron hơn.

Kết luận:

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:

1. Hu's Moments với KNN:

- **K giá trị từ 1 đến 7**:
  - **Accuracy**: Giảm dần từ 0.86 (K=1) xuống 0.816 (K=7).
  - **Precision**: Giảm từ 0.865 xuống 0.808 khi K tăng.
  - **Recall**: Dao động nhỏ, giảm từ 0.862 đến 0.815 khi K tăng.
  - **F1 score**: Từ 0.856 xuống 0.806, cho thấy khả năng cân bằng giữa precision và recall giảm khi K tăng.
- **Nhận xét**:
  - **Hu's Moments** mang lại hiệu suất tương đối ổn định khi K nhỏ. Khi K tăng, hệ thống giảm độ chính xác và sự cân bằng giữa precision và recall bị ảnh hưởng, đặc biệt ở giá trị K lớn (K=7).

2. HOG (Histogram of Oriented Gradients) với KNN:

- **K giá trị từ 1 đến 7**:
  - **Accuracy**: Giảm dần từ 0.964 (K=1) xuống 0.916 (K=7).
  - **Precision**: Từ 0.968 xuống 0.930 khi K tăng.
  - **Recall**: Từ 0.963 xuống 0.915, cho thấy khả năng phát hiện đúng mẫu giảm nhẹ khi K tăng.
  - **F1 score**: Từ 0.963 xuống 0.912, nhưng vẫn duy trì mức khá cao.
- **Nhận xét**:
  - **HOG** có hiệu suất cao hơn với các giá trị K nhỏ (K=1, K=3). Khi K tăng, các chỉ số giảm nhẹ nhưng vẫn giữ được mức độ chính xác cao so với Hu's Moments.

So sánh tổng quan:

Tiêu chí	**Hu's Moments**	**HOG**
Độ chính xác (Accuracy)	Từ 0.86 đến 0.816	Từ 0.964 đến 0.916
Độ chính xác (Precision)	Từ 0.865 đến 0.808	Từ 0.968 đến 0.930
Khả năng phát hiện (Recall)	Từ 0.862 đến 0.815	Từ 0.963 đến 0.915
F1 Score	Từ 0.856 đến 0.806	Từ 0.963 đến 0.912
Ảnh hưởng của K	Giảm dần khi K tăng	Giảm nhẹ, duy trì hiệu suất cao khi K tăng
Đặc điểm đặc trưng	Phù hợp với mẫu có hình dạng	Phù hợp với mẫu có cấu trúc phức tạp

Kết luận:

- **Hu's Moments** có hiệu suất tương đối tốt ở K nhỏ, nhưng khi K tăng, hiệu suất giảm đáng kể. Điều này cho thấy Hu's Moments có thể phù hợp hơn với các bài toán mà việc phân biệt mẫu dựa nhiều vào hình dạng.
- **HOG** mang lại hiệu suất tốt hơn ở mọi giá trị của K. Đặc biệt với K nhỏ (K=1 và K=3), **HOG** đạt độ chính xác, precision, recall, và F1 score cao hơn so với Hu's Moments. Khi K tăng, sự giảm hiệu suất không quá lớn, cho thấy khả năng phân loại của HOG ổn định hơn trong các bài toán phức tạp liên quan đến cấu trúc và hướng gradient trong ảnh.

Tóm lại, **HOG** là lựa chọn tốt hơn cho hệ thống phân loại sử dụng KNN, đặc biệt khi xử lý các dữ liệu có nhiều chi tiết hình ảnh, trong khi **Hu's Moments** có thể phù hợp với các mẫu có hình dạng đơn giản hơn.

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 0.916 đến 0.92.
  - **Precision**: Từ 0.9291 đến 0.9325, có sự ổn định và cải thiện nhẹ khi tăng số lượng neuron.
  - **Recall**: Từ 0.912 đến 0.924, cho thấy khả năng phát hiện tốt các mẫu đúng.
  - **F1 score**: Từ 0.9136 đến 0.9204, 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 không tăng quá nhiều 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 0.916 đến 0.92	Tăng dần từ 0.836 đến 0.916
Độ chính xác (Precision)	Từ 0.9291 đến 0.9325	Từ 0.7892 đến 0.9436
Khả năng phát hiện (Recall)	Từ 0.912 đến 0.924	Từ 0.836 đến 0.916
F1 Score	Từ 0.9136 đến 0.9204	Từ 0.799 đến 0.9087
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 cao 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 của bạn đã hoàn thành với các bước chính gồm:

- Thu thập dữ liệu, trích xuất đặc trưng từ hình ảnh lá cây (Hu moments và HOG).
- Sử dụng các mô hình học máy KNN và ANN để phân loại các loại lá.
- Đánh giá hiệu suất của hệ thống qua phương pháp cross-validation (5-fold).

Qua việc thử nghiệm với KNN và ANN, bạn đã có thể quan sát sự ảnh hưởng của các tham số (giá trị K của KNN, số lượng nơ-ron trong ANN) đến kết quả phân loại. Hệ thống hiện tại có thể đạt hiệu suất phân loại cao, nhưng vẫn có thể cải thiện thêm.

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

- http://elib.vku.udn.vn/bitstream/123456789/160/1/20181207154132.pdf(So sánh đặc trưng moment Hu và biểu đồ gradient có hướng trong nhận dạng tự động hoa cảnh Việt Nam Hoàng Lê Uyên Thục1 , Nguyễn Văn Đức2 , và Lê Thị Mỹ Hạnh3)
- -slides Trí Tuệ Nhân Tạo – Hoàng Lê Uyên Thục
- https://scikit-learn.org/dev/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://www.w3schools.com/python/python_ml_knn.asp
- https://www.geeksforgeeks.org/k-nearest-neighbours/
- https://viblo.asia/p/knn-k-nearest-neighbors-1-djeZ14ejKWz
- https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
- https://www.geeksforgeeks.org/artificial-neural-network-in-tensorflow/
- https://www.kaggle.com/code/srivignesh/introduction-to-ann-in-tensorflow
- https://www.tensorflow.org/tutorials/quickstart/beginner
- https://www.w3schools.com/python/python_ml_cross_validation.asp
- https://seaborn.pydata.org/generated/seaborn.heatmap.html