**ĐẠI HỌC ĐÀ NẴNG**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG**

**KHOA ĐIỆN TỬ - VIỄN THÔNG**



**BÁO CÁO CUỐI KÌ**

**CHUYÊN ĐỀ - KTMT**

*Đề tài: Phân loại 5 loại rau củ quả sử dụng HOG, PCA và SVM*

**GVHD: Trần Thị Minh Hạnh**

**Thành viên nhóm 6:**

**Huỳnh Sơn Đông 18DT3**

**Lê Chí Đức 18DT3**

**Võ Lý Hoàng Hoan 18DT3**

**Lê Văn Mạnh 18DT3**

**Phạm Thị Thanh Tuyền 18DT3**

**Đà Nẵng, 6/2022**

1. **Giới thiệu đề tài:**

Những tiến bộ gần đây trong học máy (Machine learning) đã cho phép ứng dụng rộng rãi trong mọi lĩnh vực của cuộc sống, và nông nghiệp cũng được lợi từ đó. Theo đó, Machine learning là một lĩnh vực nhỏ của AI, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể.

Thông qua dự án “Phân loại 5 loại rau củ” chúng ta sẽ hiểu hơn về AI cũng như Machine learning để thấy được sự phát triển của khoa học máy tính. Đến với dự án, ta sẽ phải tiến hành tìm kiếm hình ảnh về 5 loại trái cây. Quá trình này tốn rất nhiều thời gian và công sức nhưng bù lại nếu chúng ta thu thập càng kỹ lưỡng thì sẽ rất tốt cho việc huấn luyện máy sau này. Tập dữ liệu được xử lý, thực hiện bằng các phần mềm Photoshop, Paint… Sau đó được đưa toàn bộ lượng dữ liệu này vào để train cho máy tốn rất nhiều bộ nhớ và thời gian, từ đó ta phải tìm cách giảm chiều dữ liệu bằng PCA. Phương pháp thực hiện phân loại bằng SVM và đánh giá thông qua các đánh giá cơ bản như độ chính xác (*accuracy*), ma trận nhầm lẫn (*confusion matrix*), F1-score.

Đây là những mô hình phân loại cơ bản đã rất cũ tuy nhiên nó sẽ giúp chúng ta có cái nhìn sâu hơn về Machine learning và AI.

1. **Nội dung phương pháp:**
2. **Histogram of oriented gradients - HOG:**
3. **Định nghĩa:**

Histogram of oriented gradients (HOG) [1] là một mô tả đặc trưng được sử dụng trong thị giác máy tính và xử lý hình ảnh cho mục đích phát hiện đối tượng. Chúng ta cũng có thể sử dụng các mô tả HOG để định lượng và đại diện cho cả hình dạng và kết cấu.

1. **Các kiến thức liên quan:**

**Feature Descriptor:** Bộ mô tả đặc trưng, là một phép biến đổi dữ liệu thành các đặc trưng giúp ích cho phân loại hoặc nhận diện vật thể.

**Histogram:** Là biểu đồ histogram biểu diễn phân phối của các cường độ màu sắc theo khoảng giá trị.

**Gradient:** Là đạo hàm của véc tơ cường độ màu sắc giúp phát hiện hướng di chuyển của các vật thể trong hình ảnh.

**Local cell:** Trong thuật toán HOG, một hình ảnh được chia thành nhiều cells bởi một lưới ô vuông. Mỗi cell được gọi là một ô cục bộ.

**Local portion:** Vùng cục bộ. Là một vùng trước trích suất ra từ ô vuông trên hình ảnh. Trong phần trình bày về thuật toán thì vùng cục bộ còn được gọi là block.

**Local normalization:** Phép chuẩn hóa được thực hiện trên một vùng cục bộ. Thường là chia cho norm chuẩn bậc 2 hoặc norm chuẩn bậc 1. Mục đích của việc chuẩn hóa là để đồng nhất các giá trị cường độ màu sắc về chung một phân phối.

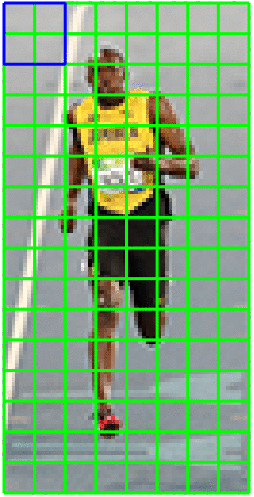
**Gradient direction:** Phương gradient. Là độ lớn góc giữa véc tơ gradient *x* và *y* giúp xác định phương thay đổi cường độ màu sắc hay chính là phương đổ bóng của hình ảnh. Giả sử *Gx*, *Gy* lần lượt là giá trị gradient theo lần lượt phương *x* và *y* của hình ảnh. Khi đó phương gradient được tính như sau:

**(1)**

**Gradient magnitude:** Độ lớn gradient. Là chiều dài của véc tơ gradient theo phương x và phương y. Biểu diễn phân phối histogram của véc tơ này theo véc tơ phương gradient sẽ thu được véc tơ mô tả đặc trưng HOG. Độ lớn gradient được tính như sau:

**(2)**

Nguyên lý hoạt động của HOG là hình dạng của một vật thể cục bộ có thể được mô tả thông qua ma trận độ lớn gradient (*gradient magnitude*) và ma trận phương gradient (*gradient direction*). Để tạo hai ma trận này, đầu tiên hình ảnh được chia thành 1 lưới ô vuông và trên đó chúng ta xác định rất nhiều các vùng cục bộ liền kề hoặc chồng lấn lên nhau. Một vùng cục bộ (*local portion*) bao gồm nhiều ô cục bộ (*local cell*) - trong thuật toán HOG là 4 - có kích thước là 8x8 pixels.

Sau đó, một biểu đồ histogram thống kê độ lớn gradient được tính toán trên mỗi ô cục bộ như trong

**Hình 2.1** nên ta sẽ có 4 véc tơ histogram kích thước ***1x9***. Bộ mô tả HOG (*HOG descriptor*) được tạo thành bằng cách nối liền (*concatenate*) 4 véc tơ histogram ứng với mỗi ô thành một véc tơ tổng hợp kích thước ***1x36***. Để cải thiện độ chính xác, mỗi giá trị của véc tơ histogram trên vùng cục bộ sẽ được chuẩn hóa theo norm chuẩn bậc 2 hoặc bậc 1. Phép chuẩn hóa này nhằm tạo ra sự bất biến tốt hơn đối với những thay đổi trong chiếu sáng và đổ bóng.

**Hình 2.1** [2]

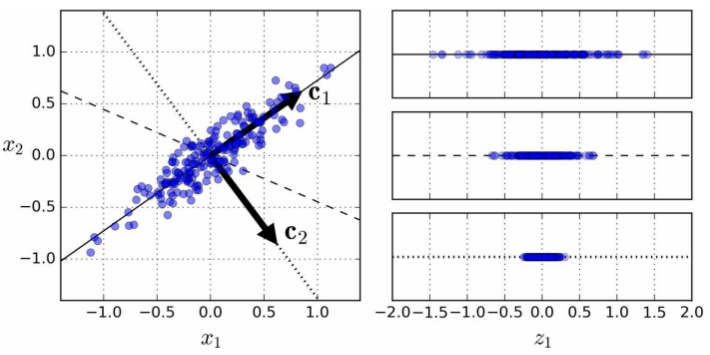
Quá trình tính toán HOG sẽ trượt 7 lần theo chiều rộng và 15 lần theo chiều cao. Ta sẽ có tổng cộng ***7x15=105*** patches, mỗi patch tương ứng với 1 véc tơ histogram kích thước ***1x36***. Do đó véc tơ HOG của ảnh sẽ có kích thước là ***105x36=3780*** chiều. Đây là một véc tơ kích thước lớn nên có thể mô phỏng được đặc trưng của ảnh khá tốt.

1. **Principal Component Analysis - PCA:**
2. **Định nghĩa:**

Principal Component Analysis (PCA) [3] cho đến nay là thuật toán giảm kích thước phổ biến nhất. Đầu tiên, nó xác định siêu phẳng nằm gần dữ liệu nhất và sau đó nó chiếu dữ liệu lên đó.

1. **Kiến thức liên quan:**

Trước khi ta có thể chiếu tập huấn luyện lên một siêu phẳng có chiều bé hơn, trước tiên ta cần phải chọn siêu phẳng phù hợp.



**Hình 2.2 Chọn không gian phụ để chiếu**[4]

Trong hình, ta có thể nhìn thấy rằng với siêu phẳng thứ nhất (đường nét liền) bảo toàn tối đa phương sai và nó mất ít thông tin hơn so với các siêu phẳng thứ hai (đường nét đứt) và thứ ba (đường nét chấm). Đồng thời trung bình bình phương khoảng cách giữa tập dữ liệu gốc và phép chiếu của nó lên trục đó cũng là ngắn nhất. PCA xác định trục chiếm lượng phương sai lớn nhất trong tập huấn luyện. Nó cũng tìm thấy trục thứ hai, trực giao với trục thứ nhất, chiếm lượng phương sai lớn nhất còn lại. Nếu tập dữ liệu có nhiều chiều hơn, PCA cũng sẽ tìm thấy các trục bằng số thứ nguyên trong tập dữ liệu. Vectơ đơn vị xác định trục được gọi là thành phần chính (*principal component* - PC). Trong **Hình 2.2**, PC1 là c1 và PC2 là c2.

Để tìm các thành phần chính của tập dữ liệu ta phải sử dụng kỹ thuật phân tích nhân tử ma trận tiêu chuẩn được gọi là Singular Value Decomposition (SVD) có thể phân rã ma trận tập huấn luyện X thành tích điểm của ba ma trận:

**(3)**

Trong đó:

**X*m x n*** là ma trận huấn luyện.

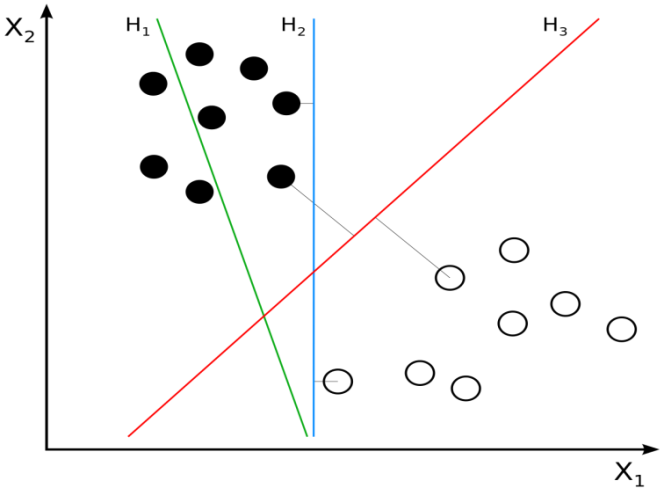
**U*m x m***, **V*n x n*** là các ma trận trực giao.

**S*m x n*** là ma trận đường chéo chữ nhật với các số thực không âm.

1. **Support vector machine - SVM:**
2. **Định nghĩa:**

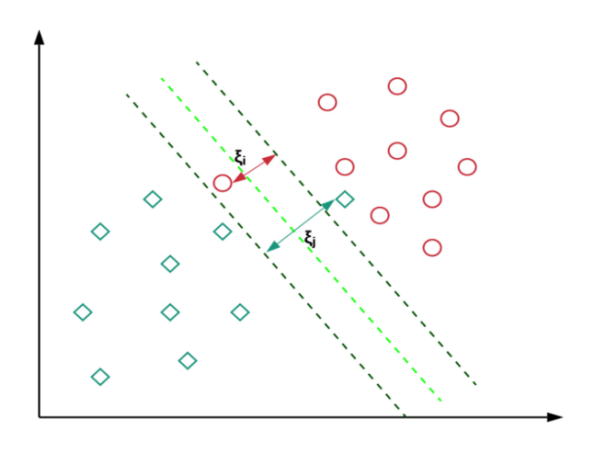
Support Vector Machine (SVM) [5] là một mô hình học máy rất mạnh mẽ và linh hoạt, có khả năng thực hiện phân loại tuyến tính hoặc phi tuyến, hồi quy và thậm chí phát hiện ngoại lệ. Đây là một trong những mô hình phổ biến nhất trong học máy. SVM đặc biệt thích hợp để phân loại các tập dữ liệu phức tạp nhưng có kích thước vừa hoặc nhỏ.

1. **Kiến thức liên quan:**



**Hình 2.3 Tập dữ liệu được phân tách bằng đường thẳng** [6]

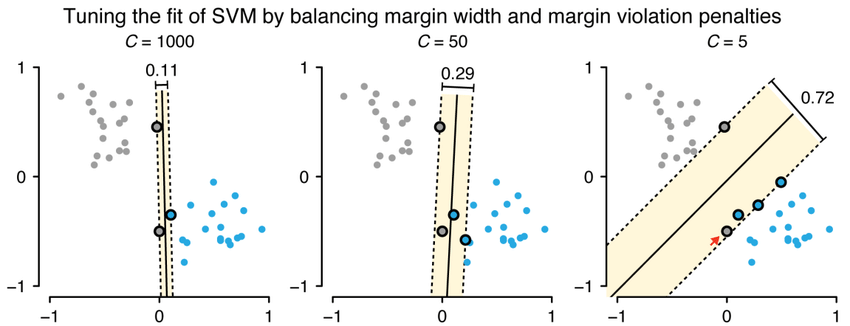
**Hình 2.3** cho thấy tập dữ liệu có thể được phân tách dễ dàng bằng một đường thẳng. Ta thấy được ranh giới quyết định của ba bộ phân loại tuyến tính có thể có. Mô hình có ranh giới quyết định được thể hiện bằng đường màu xanh lá (H1) tệ đến mức nó thậm chí không phân tách các lớp một cách chính xác. Mô hình khác có ranh giới quyết định được thể hiện bằng đường màu xanh (H2) hoạt động hoàn hảo trên tập huấn luyện này, nhưng ranh giới quyết định của nó quá gần với các trường hợp mà các mô hình này có thể sẽ không hoạt động tốt với các dữ liệu mới. Ngược lại, đường màu đỏ (H3) trong biểu đồ bên phải thể hiện ranh giới quyết định của bộ phân loại SVM. Đường thẳng này không chỉ phân tách hai lớp mà còn nằm càng xa các trường hợp huấn luyện gần nhất. Một phân loại SVM phù hợp là một phân loại với khoảng cách từ dữ liệu gần nhất đến ranh giới quyết định là rộng nhất có thể. Đây được gọi là phân loại lề lớn (*large margin classification*).



**Hình 2.4 Ảnh hưởng của phân loại lề cứng** [7]

Có hai loại phân loại lề là phân loại lề cứng (*hard margin classification*) và phân loại lề mềm (*soft margin classification*). Phân loại lề cứng là phân loại áp đặt nghiêm ngặt rằng tất cả các trường hợp cùng một lớp phải đều nằm ở cùng một phía.

Tuy nhiên với dữ liệu như **Hình 2.4** ta không thể tìm thấy được ranh giới quyết định tuyến tính với các trường hợp nhiễu. Do đó ta sẽ phải sử dụng phân loại lề mềm. Phân loại lề mềm có mục tiêu là tìm ra sự cân bằng giữa việc giữ cho lề càng rộng càng tốt và đồng thời cho phép các trường hợp vi phạm ở mức tối thiểu. Giá trị của siêu tham số C đại diện cho sự cân bằng này.



**Hình 2.5 Ảnh hưởng của tham số C** [8]

Trong **Hình 2.5**, với giá trị C cao thì bộ phân loại làm cho ít vi phạm lề hơn nhưng kết quả là làm lề nhỏ hơn. Ngược lại, sử dụng giá trị C thấp thì lề lớn hơn nhiều, nhưng nhiều trường hợp vi phạm lề hơn.

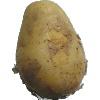
1. **Thực nghiệm và đánh giá kết quả:**
2. **Cơ sở dữ liệu:**

Cơ sở dữ liệu của tôi bao gồm 5 loại củ quả gồm eggplant (cà tím), ginger (gừng), onion (hành tây), potato (khoai tây), tomato (cà chua).

Tổng số ảnh: 2850 ảnh (train: 2140 ảnh, test: 710 ảnh).

Kích thước mỗi ảnh: 100x100 pixels.

Các củ quả được chụp 360 với tốc độ quay vừa phải, phía sau tôi đặt một tờ giấy trắng làm nền. Tuy nhiên với độ sáng khác nhau nên tôi đã tách đối tượng ra khỏi nền và thay thế nền bằng nền trắng như **Hình 3.1** .

****

**Hình 3.1 Ảnh được đưa vào mô hình huấn luyện**

1. **Khai triển Histogram of oriented gradients - HOG:**

Đầu tiên ảnh đầu vào có kích thước là 100x100 pixels được resize về kích thước 64x128 pixels. Tiếp đến ta sử dụng thư viện có sẵn để trích xuất đặc trưng theo như lý thuyết trình bày bên trên.

**from skimage.feature import hog**

**fd, hog\_image = hog( img,**

**orientations=9, pixels\_per\_cell=(8,8), cells\_per\_block=(2,2), visualize=True, channel\_axis=-1)**

Trong đó:

- fd là véc tơ đặc trưng của ảnh có kích thước là 1x3780.

- hog\_image là ảnh biểu diễn phân phối của HOG.

- img là ảnh đầu vào có kích thước 64x128 pixels.

- orientations là kích thước véc tơ histogram 1x9 được tính dựa vào ma trận độ lớn gradient (*gradient magnitude*) và ma trận phương gradient (*gradient direction*).

- pixels\_per\_cell là kích thước ô cục bộ 8x8 pixels.

- cells\_per\_block là kích thước vùng cục bộ 2x2.

Vậy với tập dữ liệu huấn luyện ta sẽ có kích thước 2850x3780 và tập dữ liệu kiểm thử sẽ có kích thước 710x3780.

1. **Khai triển Principal Component Analysis - PCA:**

Trước khi sử dụng PCA tôi đã tạo tập dữ liệu theo 2 cách đó là sử dụng trích xuất đặc trưng HOG và sử dụng véc tơ tạo nên từ giá trị của từng điểm ảnh thông qua ảnh xám có kích thước 32x32 pixels nên ta sẽ có thêm một tập dữ liệu huấn luyện và kiểm thử khác có kích thước tương ứng là 2580x1024 và 710x1024.

Bước 1: Chuẩn hóa dữ liệu:

**X = (X - X.mean()) / X.std(ddof=0)**

*\* Ta có thể sử dụng thư viện Scikit-learning để chuẩn hóa:*

**from sklearn.preprocessing import StandardScaler**

**X = StandardScaler().fit\_transform(X)**

Bước 2: Tính toán ma trận tương quan:

**X\_corr=(1/len(X)\*X.T.dot(X))**

***\**** *Ta có thể sử dụng hàm tính toán ma trận tương quan có sẵn:*

**X\_corr = X.corr()**

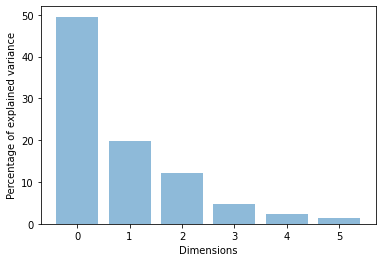
Bước 3: Sử dụng SVD để tính toán các ma trận U, S, V và tính toán dữ liệu huấn luyện sau khi giảm số chiều:

**import numpy**

**u,s,v = numpy.linalg.svd(X\_corr)**

**W = v.T[:,:d]**

Ta có thể lựa chọn tham số d là số chiều sau khi giảm. Ở đây tôi đã lựa chọn số chiều là 6 dựa vào tỉ lệ phương sai được giải thích (Explained Variance Ratio) tính toán được là 89,76%. Công thức tính:



**Hình 3.2**: **6 thành phần chính đầu tiên**

**Hình 3.2** mô tả tỉ lệ mang thông tin của 6 thành phần chính đầu tiên so với những thành phần chính khác.

Từ đó ta sẽ tính được dữ liệu huấn luyện sau khi sử dụng PCA

**X\_lower = X.dot(W)**

*\* Ta có thể sử dụng thư viện Scikit-learning để thực hiện giảm số chiều bằng PCA sau khi đã chuẩn hóa theo 2 cách:*

***Cách 1: Theo Explained Variance Ratio***

**from sklearn.decomposition import PCA**

**pca = PCA(0.9)**

***Cách 2: Theo số chiều***

**from sklearn.decomposition import PCA**

**pca = PCA(n\_components = 6)**

*Cuối cùng ta chiếu dữ liệu huấn luyện đến các thành phần chính và đưa dữ liệu vào các mô hình huấn luyện.*

**X = pca.fit\_transform(X)**

1. **Khai triển Support vector machine - SVM:**

Tôi đã cho dữ liệu đầu vào mô hình SVM theo 3 cách khác nhau. Đầu tiên là sử dụng tập dữ liệu train của HOG có kích thước 2850x3780. Cách thứ hai cũng là sử dụng tập dữ liệu train của HOG tuy nhiên tôi đã dùng PCA để giảm số chiều. Cách cuối cùng là sử dụng bộ dữ liệu từ giá trị từng điểm ảnh của ảnh xám thông qua PCA để giảm số chiều. Tuy dữ liệu huấn luyện là khác nhau nhưng ở đây tôi sử dụng chung mô hình SVM của thư viện scikit-learn có sẵn.

**from sklearn.svm import SVC**

**model = SVC(C=100,kernel = 'rbf', random\_state = 0)**

**model.fit(X\_train\_lower, y\_train.ravel())**

Trong đó:

- C là hằng số biểu diễn sự cân bằng giữa việc giữ cho lề càng rộng và đồng thời cho phép các trường hợp vi phạm ở mức tối thiểu.

- kernel là hàm toán học sử dụng trong mô hình phân loại SVM. Có thể là rbf (*radial basic function*), linear (tuyến tính), poly, ...

1. **Phương pháp đánh giá:**

Trong dự án này tôi đã sử dụng 3 phương pháp đánh giá chính là accuracy (độ chính xác), *Confusion matrix* (ma trận nhầm lẫn), *F1-score.*

1. **Accuracy (độ chính xác):**

Cách đơn giản và hay được sử dụng nhất là accuracy (độ chính xác). Cách đánh giá này đơn giản là tính tỉ lệ giữa số mẫu được dự đoán đúng và tổng số mẫu trong tập dữ liệu kiểm thử.

**(5)**

1. **Confusion matrix (ma trận nhầm lẫn):**

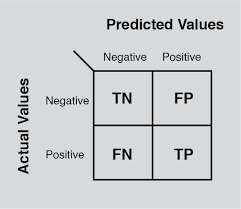
Cách tính sử dụng *Confusion matrix* (ma trận nhầm lẫn) chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, chúng ta sử dụng một ma trận được gọi là *confusion matrix*.

Về cơ bản, *confusion matrix* thể hiện có bao nhiêu mẫu dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class. Cách biểu diễn bằng màu này rất hữu ích với các bài toán với nhiều lớp dữ liệu như dự án của tôi. Một mô hình tốt sẽ cho một *confusion matrix* có các phần tử trên đường chéo chính có giá trị lớn, các phần tử còn lại có giá trị nhỏ hơn.

1. **F1-score:**

F1-score kết hợp giữa ***precision***(độ chính xác) và ***recall***(khả năng thu hồi) của bộ phân loại thành một số liệu duy nhất bằng cách lấy *harmonic mean* (trung bình hài hòa) của chúng.

(6)



**Hình 3.3 Confusion matrix** [9]

Trong **Hình 3.3**, ***precision***được định nghĩa là tỉ lệ số mẫu ***true positiv*e** trong số những điểm **được phân loại là**positive (TP + FP).

(7)

***Recall***được định nghĩa là tỉ lệ số mẫu ***true positive*** trong số những điểm **thực sự là***positive*(TP + FN).

(8)

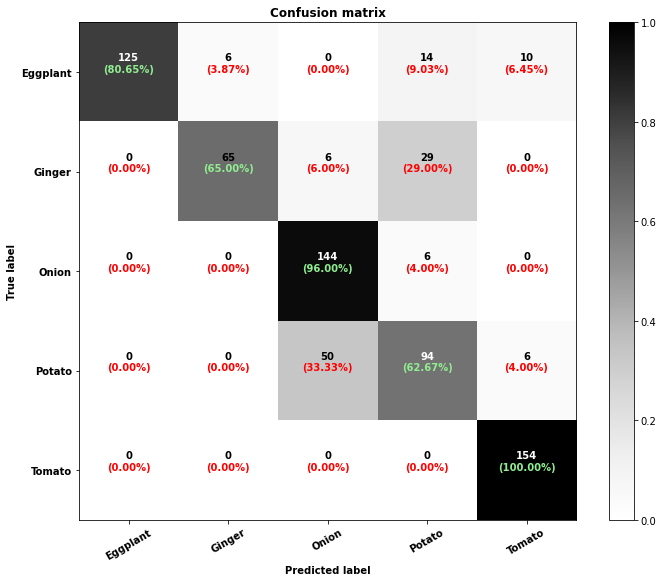
1. **Kết quả và đánh giá kết quả:**
2. **Accuracy (độ chính xác):**

|  |  |  |
| --- | --- | --- |
|  | **Sử dụng scikit-learn** | **Không sử dụng**  **scikit-learn** |
| **HOG, PCA (0.9), SVM** | 53,66% |  |
| **PCA (d=6), SVM** |  | 82,08% |
| **HOG, SVM (linear)** | 86,61% | |
| **HOG, SVM (rbf)** | 82,11% | |
| **HOG, KNN (K=3)** | 83,66% | |

***Bảng 1 Độ chính xác của một số mô hình huấn luyện***

Thông qua các mô hình huấn luyện trên ta có thể thấy được sự kết hợp giữa HOG và SVM đạt độ chính xác cao vì bản chất dữ liệu huấn luyện của HOG chứa rất nhiều đặc trưng (3780 đặc trưng) nên nó sẽ mô tả chính xác các mẫu hơn. Với mô hình PCA và SVM ta thấy được độ chính xác là tương đối, đồng thời PCA cũng đã giảm số chiều thành 6 đặc trưng nên sẽ tối ưu về bộ nhớ cho mô hình huấn luyện SVM. Với mô hình huấn luyện SVM trong dự án này thì ta thấy được độ chính xác của mô hình SVM tuyến tính cao hơn. Do đó các đánh giá sau này tôi sẽ tập trung vào PCA (d=6) và SVM.

1. **Confusion matrix (ma trận nhầm lẫn):**



**Hình 3.4 Confusion matrix của dự án**

Trong **Hình 3.4** Ta có thể thấy rằng tổng các phần tử trong toàn ma trận này chính là số mẫu trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số mẫu được phân loại đúng của mỗi lớp dữ liệu. Từ đây có thể suy ra accuracy chính bằng tổng các phần tử trên đường chéo chia cho tổng các phần tử của toàn ma trận. Cà chua (tomato) đạt độ chính xác 100% vì cơ sở dữ liệu chỉ có quả cà chua có màu đỏ nên khi lập véc tơ đặc trưng về độ sáng, màu sắc sẽ phân biệt rõ giữa cà chua và các loại khác. Gừng (ginger), hành tây (onion) và khoai tây (potato) có màu sắc tương đối giống nhau nên có sự nhầm lẫn này cũng có thể giải thích.

1. **F1-score:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** |
| **Eggplant** | 1 | 0.8065 | 0.8929 |
| **Ginger** | 0.9155 | 0.65 | 0.7602 |
| **Onion** | 0.72 | 0.96 | 0.8228 |
| **Potato** | 0.6573 | 0.6267 | 0.6416 |
| **Tomato** | 0.9059 | 1 | 0.9506 |
| ***Macro*** |  |  | ***0.8136*** |

**Bảng 2. F1-score, Precision và Recall**

1. **Kết luận:**

Sau khi thực hiện dự án, có thể thấy rõ rằng phương pháp PCA kết hợp SVM thu được hiệu suất nhận dạng tốt nhất, sau đó đến phương pháp HOG kết hợp. Thuật toán của PCA kết hợp SVM có tốc độ tiếp nhận, xử lý dữ liệu có kích thước lớn và nhanh chóng thêm vào đó là khả năng nhận dạng với độ chính xác cao. Cuối cùng là phương pháp so khớp mẫu là phương pháp thủ công, ở quy trình thực hiện, phương pháp này tuy đạt độ chính xác cao tuy nhiên tốn khá nhiều thời gian để huấn luyện. Hướng phát triển của dự án có thể kể đến như:

- Để tăng hiệu suất cho mô hình ta cần tăng số lượng và chất lượng dữ liệu đầu vào.

- Nếu ta mở rộng số lượng đối tượng nhận dạng, cải thiện tốc độ xử lý tập dữ liệu thì đây sẽ là một dự án tiềm năng.

- Tìm hiểu thêm các phương pháp trích đặc trưng khác tốt hơn để cải thiện tốc độ huấn luyện.

- Sử dụng các neural network như ANN, CNN,... tăng số lớp ẩn trong phương pháp mạng thần kinh nhân tạo nhằm nâng cao hiệu suất cũng như độ chính xác cho đầu ra.

**Tài liệu tham khảo:**

**[1]** Dalal, Navneet, and Bill Triggs.*[“Histograms of oriented gradients for human detection."](https://hal.inria.fr/file/index/docid/548512/filename/hog_cvpr2005.pdf)* Computer Vision and Pattern Recognition, 2005.

1. https://phamdinhkhanh.github.io/2019/11/22/HOG.html
2. Aurélien Géron. *“Hands-On Machine Learning with Scikit-Learn & TensorFlow”*, 2017
3. Aurélien Géron. *“Hands-On Machine Learning with Scikit-Learn & TensorFlow”*, 2017
4. <https://www.geeksforgeeks.org/support-vector-machine-algorithm/>
5. <https://en.wikipedia.org/wiki/Support-vector_machine>
6. <https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe>
7. <https://www.analyticsvidhya.com/blog/2021/04/insight-into-svm-support-vector-machine-along-with-code/>
8. <https://in2techs.com/confusion-matrix/?doing_wp_cron=1654100167.2129600048065185546875>

**Phụ lục**

1. ***Code HOG***

import os

from glob import glob

from PIL import Image

from skimage.feature import hog

import matplotlib.pyplot as plt

import csv

# reading the image

hog\_images = []

hog\_features = []

Type = ['Eggplant','Ginger','Onion','Potato','Tomato']

for i in range(len(Type)):

path\_input = './TestData/' + Type[i] + '/'

# print(path\_input)

for file in glob(path\_input + "\*.jpg"):

print(file)

images = Image.open(file)

# images.show()

width, height = images.size

print(width, height)

resized\_img = images.resize((64, 128))

widthre, heightre = resized\_img.size

# resized\_img.show()

print(widthre, heightre)

fd, hog\_image = hog(resized\_img, orientations=9, pixels\_per\_cell=(8, 8),

cells\_per\_block=(2, 2), visualize=True, channel\_axis=-1)

print(fd)

hog\_images.append(hog\_image)

hog\_features.append(fd)

import pandas as pd

pd.DataFrame(hog\_features).to\_csv('TestData.csv')

1. ***Code tạo cơ sở dữ liệu cho PCA:***

import os

from glob import glob

from PIL import Image

import numpy as np

import pandas as pd

import csv

features = []

Type = ['Eggplant','Ginger','Onion','Potato','Tomato']

for i in range(len(Type)):

path\_input = './TrainData/' + Type[i] + '/'

# print(path\_input)

for file in glob(path\_input + "\*.jpg"):

print(file)

image = Image.open(file)

imgGray = image.convert('L')

image = imgGray.resize((32, 32))

image\_sequence = image.getdata()

image\_array = []

image\_array = np.array(image\_sequence)

for i in range(len(image\_array)):

image\_array[i] = 255 - image\_array[i]

print(image\_array.shape)

features.append(image\_array)

print(len(features))

# break

with open('Data\_train.csv', 'w', newline='') as file:

mywriter = csv.writer(file, delimiter=',')

mywriter.writerows(features)

# print(features.shape)

# pd.DataFrame(image\_array).to\_csv('TestData.csv')

1. ***Code mô hình huấn luyện PCA không dùng sklearn, SVM***

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

X\_train = pd.read\_csv('/content/drive/MyDrive/CKCD/Data/Data\_train.csv')

y\_train = pd.read\_csv('/content/drive/MyDrive/CKCD/Data/TrainLabel.csv')

X\_test = pd.read\_csv('/content/drive/MyDrive/CKCD/Data/Data\_test.csv')

y\_test = pd.read\_csv('/content/drive/MyDrive/CKCD/Data/TestLabel.csv')

# X\_train = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/DimensionalityReduction/Data/TrainData.csv')

# y\_train = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/DimensionalityReduction/Data/TrainLabel.csv')

# X\_test = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/DimensionalityReduction/Data/TestData.csv')

# y\_test = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/DimensionalityReduction/Data/TestLabel.csv')

X\_train = X\_train.iloc[:,:].values

print(X\_train.shape)

y\_train = y\_train.iloc[1:,:].values

print(y\_train.shape)

X\_test = X\_test.iloc[:,:].values

print(X\_test.shape)

y\_test = y\_test.iloc[1:,:].values

print(y\_test.shape)

def standardized(X):

X = (X - X.mean()) / X.std(ddof=0)

return X

X\_train = standardized(X\_train)

X\_test = standardized(X\_test)

# print(X\_train[0:5])

# print(X\_test[0:5])

X\_train\_corr = (1 / len(X\_train) \* X\_train.T.dot(X\_train))

print(X\_train\_corr.shape)

u,s,v = np.linalg.svd(X\_train\_corr)

eig\_values, eig\_vectors = s, v.T

print(np.sum(eig\_values))

print(np.sum(eig\_values[0:6]/np.sum(eig\_values)))

print(eig\_vectors.shape)

explained\_variance=(eig\_values[0:6] / np.sum(eig\_values))\*100

plt.figure

plt.bar(range(len(eig\_vectors[0:6])), explained\_variance[0:6], alpha=0.5)

plt.ylabel('Percentage of explained variance')

plt.xlabel('Dimensions')

X\_train\_lower = X\_train.dot(eig\_vectors[:,0:6])

X\_test\_lower = X\_test.dot(eig\_vectors[:,0:6])

print(X\_train\_lower.shape)

from sklearn.svm import SVC

model = SVC(kernel = 'rbf', random\_state = 0)

model.fit(X\_train\_lower, y\_train.ravel())

y\_pred = model.predict(X\_test\_lower)

count = 0

for i in range(len(y\_pred)):

if y\_pred[i] == y\_test[i]:

count += 1

accuracy = count/len(y\_pred)

print(accuracy)

1. ***Code vẽ confusion matrix***

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix'):

fmt = 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt)+'\n', weight='bold',

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

cm = cm.astype('float') / cm.sum(axis=1, keepdims = True)

plt.imshow(cm, interpolation='nearest', cmap='Greys')

plt.title(title, weight='bold')

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=30, weight='bold')

plt.yticks(tick\_marks, classes, weight='bold')

fmt = '.2f'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, '('+format(cm[i, j]\*100, fmt)+'%)', weight='bold',

horizontalalignment="center",

color="lightgreen" if cm[i, j] > thresh else "red")

plt.tight\_layout()

plt.ylabel('True label', weight='bold')

plt.xlabel('Predicted label', weight='bold')

class\_names = ['Eggplant', 'Ginger', 'Onion', 'Potato', 'Tomato']

plt.figure(figsize=(10,8))

plot\_confusion\_matrix(cnf\_matrix, classes=class\_names,

title='Confusion matrix')

1. ***Code HOG và KNN***

import pandas as pd

import numpy as np

import math

import matplotlib.pyplot as plt

Traindata = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day8/TrainData.csv')

Trainlabel = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day8/TrainLabel.csv')

Testdata = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day8/TestData.csv')

Testlabel = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day8/TestLabel.csv')

X\_train = Traindata.iloc[:,1:].values

print(X\_train.shape)

Y\_train = Trainlabel.iloc[:,:].values

print(Y\_train.shape)

X\_test = Testdata.iloc[:,1:].values

print(X\_test.shape)

Y\_test = Testlabel.iloc[:,:].values

print(Y\_test.shape)

# print(X\_train[0:10,:])

def normalize( X ) :

X[:, 1:] = ( X[:, 1:] - np.mean( X[:, 1:], axis = 0 ) ) / np.std( X[:, 1:], axis = 0 )

return X

X\_train = normalize(X\_train)

X\_test = normalize(X\_test)

# from sklearn.model\_selection import train\_test\_split

# X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.1, random\_state = 0)

print("Total Number of rows in train:",X\_train.shape[0])

print("Total Number of rows in test:",X\_test.shape[0])

class KNN():

def \_\_init\_\_( self, K ) :

self.K = K

def fit( self, X\_train, Y\_train ) :

self.X\_train = X\_train

self.Y\_train = Y\_train

self.m\_train, self.n\_train = X\_train.shape

def predict(self, X\_test):

self.X\_test = X\_test

self.m\_test, self.n\_test = X\_test.shape

Y\_predict = np.zeros( self.m\_test)

i = 0

for i in range(self.m\_test):

x = self.X\_test[i]

neighbors = np.zeros( self.K )

neighbors = self.find\_neighbors(x)

Y\_predict[i] = self.vote(neighbors)

return Y\_predict

def vote(self,neighbors):

self.n\_nei = len(neighbors)

i=0

count1 = 0

count2 = 0

count3 = 0

count4 = 0

count5 = 0

for i in range (self.n\_nei):

if neighbors[i] == 1:

count1 += 1

elif neighbors[i] == 2 :

count2 += 1

elif neighbors[i] == 3 :

count3 += 1

elif neighbors[i] == 4 :

count4 += 1

else:

count5 += 1

if (count1>=count2) & (count1>=count3) & (count1>=count4) & (count1>=count5):

return 1

elif (count2>=count1) & (count2>=count3) & (count2>=count4) & (count2>=count5):

return 2

elif (count3>=count2) & (count3>=count1) & (count3>=count4) & (count3>=count5):

return 3

elif (count4>=count1) & (count4>=count2) & (count4>=count3) & (count4>=count5):

return 4

else:

return 5

def find\_neighbors(self, x):

euclidean\_distances = np.zeros(self.m\_train )

for i in range( self.m\_train ) :

d = self.euclidean( x, self.X\_train[i] )

euclidean\_distances[i] = d

inds = euclidean\_distances.argsort()

Y\_train\_sorted = self.Y\_train[inds]

return Y\_train\_sorted[:self.K]

def euclidean( self, x, X\_train ) :

return np.sqrt( np.sum( np.square( x - X\_train ) ) )

model = KNN(K=3)

model.fit(X\_train,Y\_train)

y\_pred = model.predict(X\_test)

print(y\_pred)

accuracy = 0

for i in range(len(y\_pred)):

if y\_pred[i] == Y\_test[i]:

accuracy += 1

print(f"Accuracy = {accuracy / len(y\_pred)}")

1. ***Code PCA sử dụng sklearn, SVM***

import pandas as pd

import numpy as np

X\_train = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day10/Data\_Train.csv')

y\_train = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day10/TrainLabel.csv')

X\_test = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day10/Data\_Test.csv')

y\_test = pd.read\_csv('/content/drive/MyDrive/Colab Notebooks/MachineLearning/Day10/TestLabel.csv')

X\_train = X\_train.iloc[:,1:].values

print(X\_train.shape)

y\_train = y\_train.iloc[:,:].values

print(y\_train.shape)

X\_test = X\_test.iloc[:,1:].values

print(X\_test.shape)

y\_test = y\_test.iloc[:,:].values

print(y\_test.shape)

from sklearn.decomposition import PCA

from sklearn.svm import SVC

def PCASVM(nc):

pca = PCA(nc)

X\_train\_lower = pca.fit\_transform(X\_train)

# print(pca.n\_components\_)

n\_components = pca.n\_components\_

PCA\_test = PCA(n\_components)

X\_test\_lower = PCA\_test.fit\_transform(X\_test)

X\_test\_lower.shape

model = SVC(kernel = 'rbf', random\_state = 0)

model.fit(X\_train\_lower, y\_train)

y\_pred = model.predict(X\_test\_lower)

return y\_pred

from sklearn.metrics import confusion\_matrix, f1\_score

# cm = confusion\_matrix(y\_test, y\_pred)

# print(cm)

nc = np.arange(0.6, 1, 0.05)

f1\_sc = []

Accuracy = []

num = 0.9

y\_pred = PCASVM(num)

count = 0

for i in range(len(y\_pred)):

if y\_pred[i] == y\_test[i]:

count += 1

accuracy = count/len(y\_pred)

Accuracy.append(accuracy)

f1 = f1\_score(y\_test, y\_pred, average='weighted')

f1\_sc.append(f1)

print(accuracy)

1. ***Cơ sở dữ liệu dạng file csv***

<https://drive.google.com/drive/folders/10_hPghvP-klAFGrM3UW3HrvxfaNbuY8X?usp=sharing>