Trường Đại học Công nghệ Thông tin

Khoa Khoa học máy tính

ĐỒ ÁN MÔN HỌC

Máy học trong thị giác máy tính

Đề tài: Nhận dạng trái cây

Author:

Đặng Hoàng Sang

Trần Hoàng Huy

Teacher:

Mai Tiến Dũng

Tp.HCM 2019-2020

Contents

1

Giới thiệu

1.1 Mô tả bài toán

Sự phát triển của công nghệ và bùng nổ về dữ liệu trong những năm trở lại đây chính là nguồn động lực to lớn thúc đẩy quá trình phát triển của học máy. Cụ thể, với lượng dữ liệu dồi dào cùng những phương pháp học cải tiến đã tăng hiệu quả của quá trình học máy, cải thiện độ chính xác cũng như thời gian huấn luyện mô hình.

Một trong những bài toán nổi tiếng, phổ biến và áp dụng rộng rãi hiện nay không thể không kể đến đó là bài toán nhận dạng (Object Recognition).

Bài toán nhận dạng đang là một đề tài thảo luận cực kì sôi nổi về khả năng và ứng dụng mà nó có thể mang lại cho cuộc sống. Nhiều nhà nghiên cứu cũng như rất nhiều dự án đã được triển khai nhằm để nghiên cứu và phát triển bài toán với hi vọng có thể khai thác hiệu quả nhất có thể của những ứng dụng của bài toán nhận diện để phục vụ cho con người.

Các bài toán ứng dụng của nhận dạng ta có thể kể đến như: Nhận dạng khuôn mặt, nhận dạng biển số xe, nhận dạng các tín hiệu giao thông, nhận dạng biểu hiện cảm xúc của con người, nhận dạng cử chỉ, hành vi,...

Bằng những dữ liệu thu thập được (dữ liệu hình ảnh, video,...) Chúng ta có thể huấn luyện, dạy cho máy tính về những đặc trưng, nội dung mà dữ liệu thể hiện với mong muốn khi gặp lại các dữ liệu tương đồng, máy tính có thể biết được những dữ liệu đó đang đề cập đến nội dung gì thông qua những cách tách xuất dữ liệu cũng như quá trình con người huấn luyện cho nó.

Thị giác máy tính (Computer Vision) là một lĩnh vực trong Artificial Intelligence và Computer Science (Trí tuệ nhân tạo và Khoa học máy tính) nhằm giúp máy tính có được khả năng nhìn và hiểu giống như con người. Hay đơn giản hơn, thị giác máy tính là một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh. Nhận dạng trái cây là một chương trình

ứng dụng để phân loại, nhận dạng các loại trái cây hoặc rau củ phổ biến từ những đặc trưng cơ bản của chúng về màu sắc (color), kích thước hình dạng (shape),... Thông qua quá trình huấn luyện cho máy tính với mục đích cho máy có khả năng nhận biết được các loại quả khác nhau.
Giới thiệu bài toán:
Input: Một bức ảnh (có thể tải lên từ thiết bị hoặc chụp từ camera của ứng dụng) trong đó có chứa 1 trái cây hoặc rau củ.
Output: Bức ảnh gốc kèm theo tên (tiếng việt và tiếng anh) của các loại trái cây rau củ có trong ảnh.

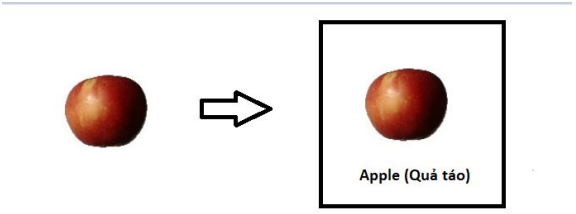

Figure 1: Ví dụ mẫu

1.2 Tầm quan trọng của bài toán

Nhận dạng trái cây là một trong những bài toán cơ bản và phù hợp cho người mới tiếp cận Học máy trong thị giác máy tính. Đây cũng có thể được coi là 1 bài toán hiệu quả để nghiên cứu, so sánh và phát triển những kĩ thuật học máy vì nó sở hữu một bộ data dễ tiếp cận và thực hành trên chúng. Ứng dụng: Nhận dạng trái cây có thể ứng dụng trong trong công tác giáo dục giúp trẻ em có thể nhận biết các được các loại trái cây và tên gọi của chúng thông qua các hình ảnh, rèn luyện khả năng tự học cho trẻ. Ứng dụng trong robot thu hoạch trái cây hoặc trong cái hệ thống tính tiền tự động trong bán hàng. Những cải tiến cho bài toán nhận dạng trái cây có thể ứng dụng để nhận biết trái cây đã chín hay chưa, trái cây có còn tươi hay không, hay hàm lượng chất dinh dưỡng trong trái cây như thế nào.

1.3 Những điểm nổi bật của phương pháp

Phương pháp của mình có những điểm nổi bật gì, điểm khác gì so với các phương pháp khác, kết quả của mình có gì vượt trội.

2

Các nghiên cứu liên quan

Công nghệ ngày càng tiến bộ cùng với sự tăng nhanh về số lượng của dữ liệu đã giúp các phương pháp học máy cải thiện được độ hiệu quả và có nhiều những ứng dụng hơn trong cuộc sống. Máy học trong thị giác máy tính là sự kết hợp giữa khả năng xử lí trên dữ liệu hình ảnh, video,... của máy tính cùng với các phương pháp học máy để phục vụ cho bài toán phân loại hay nhận diện dựa trên hình ảnh, video,.... Các bài toán nổi tiếng và ứng dụng rộng rãi hiện nay như nhận dạng khuôn mặt, nhận dạng cử chỉ, hành vi,...

Bài toán nhận dạng trái cây là bài toán khá phổ biến và cơ bản. Bộ dữ liệu nổi tiếng cho bài toán này có thể được tìm thấy ở đường dẫn: [1], bộ dữ liệu được thực hiện bởi Mihai Oltean và Horea Muresan thuộc Viện Công nghệ Massachusetts. Bài toán nhận dạng trái cây từng được nhắc đến khá lâu ở bài báo ”Fruit Recognition using Color and Texture Features” của nhóm tác giả S.Arivazhagan, R.Newlin Shebiah, S.Selva Nidhyanandhan, L.Ganesan [2] vào tháng 10 năm 2010, đưa ra cách để trích xuất đặc trưng từ ảnh trái cây dựa trên color histogram và haralick. Trong bài báo ”Image Classification using Python and Scikit-learn” của Gogul Ilango [3] có đề cập đến việc có thể nhận dạng vật thể bằng cách kết hợp 3 phương pháp trích xuất đặc trưng từ ảnh bao gồm: color histogram, hu moments và haralick cùng với phương pháp học máy Ramdom Forest. Trong bài đăng ”Computer Vision: Fruit Recognition” của tác giả Nadya Aditama [4] có đề xuất phương pháp nhận dạng trái cây bằng cách trích xuất đặc trưng Histogram of Oriented Gradient từ ảnh cùng với phương pháp học là K-Nearest Neighbor. Trong bài báo ”Fruit recognition from images using deep learning” của tác giả Mihai Oltean và Horea Muresan[5] đã sử dụng mô hình mạng học sâu deep learning để phân loại các loại trái cây. Trong các bài viết ”Fruit image recognition” tác giả Riad Shaltaf[6], ”CNN from scratch with 98% accuracy” của tác giả Anindita Pani[7] và ”Fruits-360 - Transfer Learning using Keras” của tác giả IS[8] sử dụng mô hình mạng học sâu để phân loại cho trái cây. Trong bài viết ”CNN Architectures: VGG, ResNet,

Inception + TL” của tác giả Shivam Bansal có đề xuất các kiến trúc hiện đại có thể được dùng cho bài toán phân loại trái cây như VGG16, VGG19,...

3

Phương pháp đề xuất

3.1 Giới thiệu

Nhận dạng trái cây được thực hiện bằng cách cho hệ thống học ra một mô hình để sử dụng cho việc phân loại các loại trái cây dựa trên vector đặc trưng của ảnh chứa trái cây đó được cung cấp từ training set.
Các phương pháp lấy vector đặt trưng từ ảnh được sử dụng: Histogram of Oriented Gradients, Color Histogram, Haralick, Hu Moments.
Các phương pháp học được sử dụng: K-Nearest Neightbor (KNN), Support vector machine (SVM), Convolutional Neural Network (CNN).

3.2 Các phương pháp trích xuất đặc trưng

3.2.1 Hu Moments

- Hu Moments (hình dạng) là một Image Descriptor dựa trên thuật toán Moments, sử dụng các phép thống kê để mô tả hình dạng của một đối tượng có trong bức ảnh nhị phân hoặc edged-image. Hu Moments Image Descriptor trả về một Feature Vector gồm 7 giá trị. Feature Vector này sẽ được so sánh với nhau để xác định sự tương đồng giữa hai vật thể.
- Hu Moments được giới thiệu lần đầu vào năm 1962 bởi Hu, ông đã định nghĩa ra 7 công thức dùng để tính 7 giá trị trả về của Hu Moments.
- Các công thức để tìm 7 giá trị của Hu Moments tương đối phức tạp. Trong thư viện Opencv có hỗ trợ công cụ để tính các giá trị này một cách nhanh chóng.
Công thức xác định moment như sau:

$$M_{pq} = \int_{a1}^{a2} \int_{b1}^{b2} x^p y^q f(x,y) dxdy$$

Trong đó:

•p, q = 0, 1, 2,...
•f(x,y) là giá trị tại pixel có tọa độ (x,y).
Moment được phân biệt nhờ vào 2 tham số p và q, giá trị (p+q) là thứ tự của moment đó và được ký hiệu là $M_{p,q}$. Cụ thể một số moment được định nghĩa như sau:
•$m_{0,0}$ được gọi là zero order moment, moment này chính là diện tích của đối tượng.
•$m_{0,1}$ hay $m_{1,0}$ được gọi là first order moments, contain information about the center of gravity of the object.
•$m_{2,0}$, $m_{0,2}$, $m_{1,1}$ được gọi là second order moments.
•...

Các central moment được định nghĩa như sau:
$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\bar{x})^p(y-\bar{y})^q f(x,y)dxdy$$

Với:
•p,q = 0, 1, 2,...

$$\bar{x} = \frac{M_{10}}{M_{00}}$$
$$\bar{y} = \frac{M_{01}}{M_{00}}$$

Chuẩn hóa các central moment:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}}$$
$$\gamma = \frac{p+q+2}{2}, p+q = 2,3,...$$

7 công thức Hu Moment:
$$h_1 = \eta_{20} + \eta_{02}$$
$$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$h_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \mu_{03})^2$$
$$h_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \mu_{03})^2$$
$$h_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]$$

$$+ (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$
$$h_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$
$$h_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]$$
$$- (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \mu_{03})^2]$$

3.2.2 Haralick Texture

- Haralick Texture (kết cấu – hoa văn) được dùng để mô tả kết cấu (texture) và hoa văn (pattern) của một bức ảnh, đối tượng, bao gồm vẻ bề ngoài (appearance), sự nhất quán (consistency) và cảm giác về bề mặt ("feeling of surface") có trong bức ảnh.
- Haralick Texture Features được Haralick và các cộng sự đề xuất năm 1973, tính toán bằng cách sử dụng Gray-Level Co-occurrence Matrix (GLCM). Ma trận này đặc trưng cho kết cấu của bức ảnh bằng cách ghi lại tần suất xuất hiện trong bức ảnh của các cặp điểm ảnh liền kề với các giá trị cụ thể.
- Có 4 ma trận GLCM được sử dụng để tính toán, từ mỗi ma trận sẽ cho ra 13 giá trị của Haralick Texture Featues (tức là feature shape = (4, 13)), trên thực tế là 14 nhưng giá trị thứ 14 tính toán khá phức tạp và tốn nhiều thời gian nên người ta thường chỉ tính 13 giá trị, từ 13 giá trị tìm được người ta tiếp tục tìm ra 13 giá trị trung bình từ list này. Các giá trị này sẽ được dùng để mô tả sự tương phản (contrast), tương quan (correlation), sự khác biệt (dissimilarity), entropy, tính đồng nhất (homogeneity) và các đặc tính thống kê khác.

Textural Features

1) Angular Second Moment:

$$f_1 = \sum_i \sum_j p(i,j)^2$$

2) Contrast:

$$f_2 = \sum_{n=0}^{N_{g-1}} n^2 \{ \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j) \}$$

3) Correlation:

$$f_3 = \frac{\sum_i \sum_j (ij) p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y}$$

where μ_x,μ_y,σ_x and σ_y are the means and standard deviations of p_x and p_y.
4) Sum of Squares: Variance

$$f_4 = \sum_i \sum_j (i - \mu)^2 p(i,j)$$

5) Inverse Difference Moment:

$$f_5 = \sum_i \sum_j \frac{1}{1 + (i+j)^2} p(i,j)$$

6) Sum Average:

$$f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i)$$

7) Sum Variance:

$$f_7 = \sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i)$$

8) Sum Entropy:[2]

$$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) log\{p_{x+y}(i)\}$$

9) Entropy:

$$f_9 = -\sum_i \sum_j p(i,j) log(p(i,j))$$

10) Difference Variance:

$$f_{10} = \text{variance of } p_{x-y}$$

11) Difference Entropy:

$$f_{11} = -\sum_{i=0}^{N_g-1} p_{x-y}(i) log\{p_{x-y}(i)\}$$

12), 13) Information Measures of Correlation:

$$f_{12} = \frac{HXY - HXY1}{max\{HX, HY\}}$$

$$f_{13} = (1 - exp[-2.0(HXY2 - HXY)])^{1/2}$$

$$HXY = -\sum_i \sum_j p(i,j) log(p(i,j))$$

where *HX* and *HY* are entropies of p_x and p_y and

$$HXY1 = -\sum_i \sum_j p(i,j) log\{p_x(i)p_y(j)\}$$

$$HXY2 = -\sum_i \sum_j p_x(i)p_y(j) log\{p_x(i)p_y(j)\}$$

14) Maximal Correlation Coeffident:

$$f_{14} = (\text{Second largest eigenvalue of } Q)^{1/2}$$

where

$$Q(i,j) = \sum_k \frac{p(i,K)p(j,k)}{p_x(i)p_y(k)}$$

3.2.3 Color Histogram

- Color Histogram (màu sắc) là một dạng đặc trưng toàn cục biểu diễn phân phối của các màu trên ảnh. Color Histogram thống kê số lượng các pixel có giá trị nằm trong một khoảng màu nhất định cho trước. Color histogram có thể tính trên các dạng ảnh RGB hoặc HSV, thông dụng là HSV (Hue – vùng màu, Saturation – độ bão hòa màu, Value – độ sáng).

3.2.4 Histogram of Oriented Gradient

- HOG là viết tắt của Histogram of Oriented Gradient - một loại "feature descriptor". Mục đích của "feature descriptor" là trừu tượng hóa đối tượng bằng cách trích xuất ra những đặc trưng của đối tượng đó và bỏ đi những thông tin không hữu ích. Vì vậy, HOG được sử dụng chủ yếu để mô tả hình dạng và sự xuất hiện của một đối tượng trong ảnh.Thu thập tất cả các biểu đồ cường độ gradient định hướng để tạo ra feature vector cuối cùng. Các bước thực hiện tính toán HOG từ ảnh: Chuẩn hóa hình ảnh trước khi xử lý (tiền xử lý), tính toán gradient theo cả hướng x và y, tìm vector đặc trưng cho từng ô, chuẩn hóa các block, tính toán HOG.

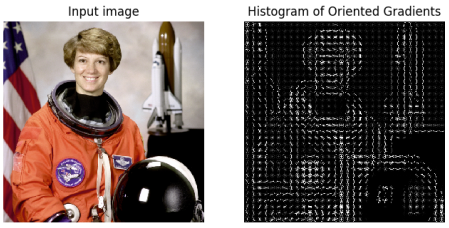

Figure 2: Minh họa Histogram of oriented gradient của 1 ảnh

3.3 Các mô hình học máy

3.3.1 K-Nearest Neighbor (KNN)

K-Nearest Neighbor là một trong những thuật toán Supervised learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi

nó cần dự đoán kết quả của dữ liệu mới, lớp (nhãn) của một đối tượng dữ liệu mới có thể dự đoán từ các lớp (nhãn) của K hàng xóm gần nó nhất.
K-Nearest Neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression.

Công thức tính khoảng cách dùng trong KNN là khoảng cách Minkowski, dạng tổng quát của Minkowski:

$$D = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$$

Với
p = 1, Manhattan Distance
p = 2, Euclidean Distance
p = ∞, Chebyshev Distance
Khoảng cách Manhattan giữa hai điểm: P1 có tọa độ (x1,y1) và điểm P2 có tọa độ (x2,y2) là $|x1 - x2| + |y1 - y2|$
Khoảng cách Euclid giữa hai điểm **p** và **q** là chiều dài đoạn thẳng . Trong hệ tọa độ Descartes, nếu **p** = $(p_1, p_2, ..., p_n)$ và **q** = $(q_1, q_2, ..., q_n)$ là hai điểm trong không gian Euclid n chiều, thì khoảng cách từ **p** đến **q** bằng:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Khoảng cách Chebyshev giữa hai vector **p** và **q**, với tọa độ p_i và q_i là:

$$D_{Chebyshev}(p, q) := max_i(|q_i - p_i|^2)$$

Bằng giới hạn không gian của L_p metrics:

$$\lim_{k \to \infty} (\sum_{i=1}^{n} |p_i - q_i|^k)^{1/k}$$

3.3.2 Support Vector Machine (SVM)

SVM là một thuật toán giám sát, nó có thể sử dụng cho cả việc phân loại hoặc đệ quy. Tuy nhiên, nó được sử dụng chủ yếu cho việc phân loại. Nó không chỉ hoạt động tốt với các dữ liệu được phân tách tuyến tính mà còn tốt với cả dữ liệu phân tách phi tuyến. Với nhiều bài toán, SVM mang lại kết quả tốt như mạng nơ-ron với hiệu quả sử dụng tài nguyên tốt hơn hẳn. SVM sử dụng để tìm ra một siêu phẳng (hyperplane), mục đích của siêu phẳng đó là phân tách tập dữ liệu thành hai phần riêng biệt - tư tưởng của bài toán phân lớp.

Khoảng cách từ một điểm (vector) có tọa độ x_0 tới siêu mặt phẳng (hyperplane) có phương trình $\mathbf{w}^T\mathbf{x} + b = 0$ được xác định bởi:

$$\frac{|\mathbf{w}^T\mathbf{x}_0 + b|}{\|\mathbf{w}\|_2}$$

Với $\|w\|_2 = \sqrt{\sum_{i=1}^{d} w_i^2}$ với d là số chiều của không gian.
Với cặp dữ liệu (x_n, y_n) bất kỳ, khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|w\|_2}$$

Margin được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó (bất kỳ điểm nào trong hai classes):

$$margin = min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|w\|_2}$$

Bài toán tối ưu trong SVM chính là bài toán tìm **w** và b sao cho *margin* này đạt giá trị lớn nhất:

$$(\mathbf{w}, b) = argmax_{\mathbf{w},b} \left\{ min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|w\|_2} \right\} = argmax_{\mathbf{w},b} \left\{ \frac{1}{\|w\|_2} min_n y_n(\mathbf{w}^T\mathbf{x}_n + b) \right\}$$

Khi dùng mô hình SVM hỗ trợ bởi thư viện Sklearn, đối với bài toán multiple classification người ta thường sử dụng các Kernel để phân lớp. Các kernel phổ biến: Linear, Rbf, Poly,...Ngoài ra người ta còn quan tâm đến các hệ số như C, gramma(đối với Rbf) ,... Trong bài báo cáo này sử dụng 2 kernel để phân lớp là linear và rbf, các hệ số C và Gramma được sử dụng mặc định: C = 1 , gramma =

$1/n_{feature}$

3.3.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNNs – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến. Nó giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay. Như hệ thống xử lý ảnh lớn như Facebook, Google hay Amazon đã đưa vào sản phẩm của mình những chức năng thông minh như nhận dạng khuôn mặt người dùng, phát triển xe hơi tự lái hay drone giao hàng tự động. CNN được sử dụng nhiều trong các bài toán nhận dạng các object trong ảnh. Mạng CNN là một tập hợp các lớp convolution chồng lên nhau và sử dụng các hàm nonlinear activation như ReLU và tanh để kích hoạt các trọng số trong các node. Mỗi một lớp sau khi thông qua các hàm kích hoạt sẽ tạo ra các thông tin trừu tượng hơn cho các lớp tiếp theo.

Lớp tích chập (Convolutional layers): Lớp tích chập được dùng để phát hiện và trích xuất đặc trưng - chi tiết của ảnh. Giống như các lớp ẩn khác, lớp tích chập lấy dữ liệu đầu vào, thực hiện các phép chuyển đổi để tạo ra dữ liệu đầu vào cho lớp kế tiếp (đầu ra của lớp này là đầu vào của lớp sau). Phép biến đổi được sử dụng là phép tính tích chập. Mỗi lớp tích chập chứa một hoặc nhiều bộ lọc - bộ phát hiện đặc trưng (filter - feature detector) cho phép phát hiện và trích xuất những đặc trưng khác nhau của ảnh. Sau khi đưa một bức ảnh đầu vào cho lớp Tích chập, ta nhận được kết quả đầu ra là một loạt ảnh tương ứng với các bộ lọc đã được sử dụng để thực hiện phép tích chập. Các trọng số của các bộ lọc này được khởi tạo ngẫu nhiên trong lần đầu tiên và sẽ được cải thiện dần xuyên suốt quá trình huấn luyện. Độ phức tạp của đặc trưng được phát hiện bởi bộ lọc tỉ lệ thuận với độ sâu của lớp tích chập mà nó thuộc về. Trong mạng CNN, những lớp tích chập đầu tiên sử dụng bộ lọc hình học (geometric filters) để phát hiện những đặc trưng đơn giản như cạnh ngang, dọc, chéo của bức ảnh. Những lớp tích chập sau đó được dùng để phát hiện đối tượng nhỏ, bán hoàn chỉnh như mắt, mũi, tóc, v.v. Những lớp tích chập sâu nhất dùng để phát hiện đối tượng hoàn hỉnh như: chó, mèo, chim, ô tô, đèn giao thông, v.v.

Lớp kích hoạt phi tuyến: Lớp này được xây dựng với ý nghĩa đảm bảo tính phi tuyến của mô hình huấn luyện sau khi đã thực hiện một loạt các phép tính toán tuyến tính qua các lớp Tích chập. Lớp Kích hoạt phi tuyến nói chung sử dụng các hàm kích hoạt phi tuyến như ReLU hoặc sigmoid, tanh,... để giới hạn phạm vi biên độ cho phép của giá trị đầu ra. Trong số các hàm kích hoạt này, hàm ReLU

được chọn do cài đặt đơn giản, tốc độ xử lý nhanh mà vẫn đảm bảo được tính toán hiệu quả. Cụ thể, phép tính toán của hàm ReLU chỉ đơn giản là chuyển tất cả các giá trị âm thành giá trị 0. Thông thường, lớp Kích hoạt phi tuyến được áp dụng ngay phía sau lớp Tích chập, với đầu ra là một ảnh mới có kích thước giống với ảnh đầu vào, các giá trị điểm ảnh cũng hoàn toàn tương tự trừ các giá trị âm đã bị loại bỏ.

Lớp Pooling: Mục đích của pooling rất đơn giản, nó làm giảm số hyperparameter mà ta cần phải tính toán, từ đó giảm thời gian tính toán, tránh overfitting. Loại pooling ta thường gặp nhất là max pooling, lấy giá trị lớn nhất trong một pooling window. Pooling hoạt động gần giống với convolution, nó cũng có 1 cửa sổ trượt gọi là pooling window, cửa sổ này trượt qua từng giá trị của ma trận dữ liệu đầu vào (thường là các feature map trong convolutional layers), chọn ra một giá trị từ các gía trị nằm trong cửa sổ trượt (với max pooling ta sẽ lấy giá trị lớn nhất). Hãy cùng nhìn vào ví dụ sau, tôi chọn pooling window có kích thước là 2 * 2, stride = 2 để đảm bảo không trùng nhau, và áp dụng max pooling. Qua lớp Max Pooling thì số lượng neuron giảm đi phân nửa. Trong một mạng CNN có nhiều Feature Map nên mỗi Feature Map chúng ta sẽ cho mỗi Max Pooling khác nhau. Chúng ta có thể thấy rằng Max Pooling là cách hỏi xem trong các đặc trưng này thì đặc trưng nào là đặc trưng nhất. Ngoài Max Pooling còn có L2 Pooling. Lớp kết nối đầy đủ (Fully Connected): Lớp kết nối đầy đủ này được thiết kế hoàn toàn tương tự như trong mạng nơ-ron truyền thống tức là layer này cũng chính là 1 fully connected ANN, theo đó tất cả các điểm ảnh được kết nối đầy đủ với node trong lớp tiếp theo. So với mạng nơ-ron truyền thống, các ảnh đầu vào của lớp này đã có kích thước được giảm bớt rất nhiều, đồng thời vẫn đảm bảo các thông tin quan trọng cho việc nhận dạng. Do vậy, việc tính toán nhận dạng sử dụng mô hình truyền thẳng đã không còn phức tạp và tốn nhiều thời gian như trong mạng nơron truyền thống.

3.4 Các thư viện hỗ trợ

3.4.1 Thư viện Scikit-learn (Sklearn)

Scikit-Learn là một thư viện học máy được xây dựng dựa trên việc sử dụng các gói Python có sẵn như: gói hỗ trợ tính toán các hàm phức tạp và đại số NumPy, gói hỗ trợ các tác vụ tính toán khoa học SciPy và gói xử lý minh họa đồ thị Matplotlib. Scikit-Learn là gói thư viện bao gồm các công cụ có thể sử dụng cho các tác vụ

học máy tiêu chuẩn như phân cụm, phân lớp, hồi quy,... Dù chỉ hỗ trợ các tác vụ cơ bản, nhưng nó rất đầy đủ cho những người mới bắt đầu nghiên cứu. Nó có thể được sử dụng như một ứng dụng riêng lẻ hoặc tích hợp vào các ứng dụng khác. Tính linh hoạt khiến nó trở thành một trong các công cụ phổ biến nhất được biết đến. Một ưu điểm nữa là Scikit-Learn hoàn toàn là mã nguồn mở được cung cấp dưới dạng BSD license, do đó được phép tùy chỉnh và sử dụng lại một cách dễ dàng.

3.4.2 Thư viện Tensorflow

TensorFlow là một thư viện open source được phát triển bởi Google và công khai cho cộng đồng lập trình viên từ năm 2015. Nền tảng được nhóm Google Brain phát triển nhằm phục vụ việc tìm kiếm hình ảnh, nhận biết giọng nói và các dấu vết dữ liệu.
Với TensorFlow, Machine Learning đang trở nên gần gũi và dễ dàng tiếp cận hơn cho lập trình viên. Một số đặc điểm đáng lưu ý của thư viện này: - Core là C++ để đem lại performance cao.
- Chạy được trên cả Android IOS và Cloud.
- Hỗ trợ GPU.
- Hỗ trợ Distributed training.
- Rất nhiều ngôn ngữ front-end, nhiều nhất là Python hoặc C++
Một số project nổi tiếng sử dụng thư viện Tensorflow: - Phân loại ung thư da – Dermatologist-level classification of skin cancer with deep neural networks (Esteva et al., Nature 2017) - WaveNet: Text to speech – Wavenet: A generative model for raw audio (Oord et al., 2016) - Vẽ hình – Draw Together with a Neural Network (Ha et al., 2017) - Image Style Transfer Using Convolutional Neural Networks (Gatys et al., 2016) Tensorflow adaptation by Cameroon Smith (cysmith@github)

3.4.3 Thư viện Keras

Tổng quan Keras là một library được phát triển vào năm 2015 bởi François Chollet, là một kỹ sư nghiên cứu Deep Learning tại Google. Nó là một open source cho neural network được viết bởi ngôn ngữ python. Keras là một API bậc cao có thể sử dụng chung với các thư viện Deep Learning nổi tiếng như TensorFlow (được phát triển bởi Google), CNTK (được phát triển bởi Microsoft), Theano

(người phát triển chính Yoshua Bengio). Keras được coi là một thư viện 'high-level' với phần 'low-level' (còn được gọi là back-end) có thể là TensorFlow, CNTK, hoặc Theano. Keras có cú pháp đơn giản hơn TensorFlow rất nhiều. Keras có một số ưu điểm như:
- Dễ sử dụng, xây dựng model nhanh.
- Có thể hoạt động trên cả CPU và GPU.
- Hỗ trợ xây dựng CNN, RNN và có thể kết hợp cả 2.
- Hỗ trợ tính toán với GPU và các hệ thống phân tán. Điều này là tối quan trọng vì việc huấn luyện các mô hình Deep Learning yêu cầu khả năng tính toán rất mạnh.
- Hỗ trợ các ngôn ngữ lập trình phổ biến: C/C++, Python, Java, R,....
- Có thể chạy được trên nhiều hệ điều hành.
- Thời gian từ ý tưởng tới xây dựng và huấn luyện mô hình ngắn.
- Có thể chạy trên trình duyệt và các thiết bị di động.
- Có khả năng giúp người lập trình can thiệp sâu vào mô hình và tạo ra các mô hình phức tạp.
- Chứa nhiều model zoo, tức các mô hình Deep Learning thông dụng đã được huấn luyện.
- Hỗ trợ tính toán backpropagation tự động.
- Có cộng đồng hỏi đáp lớn.

Keras core modules Keras layers: Trong phần này, mình sẽ nói về 3 kiểu layers chính trong Keras ứng với 3 kiến trúc mạng nơ-ron phổ biến hiện tại: Deep Neural Network (DNN), Convolutional Neural Network (CNN), và Recurrent Neural Network (RNN).
1) DNN layers: DNN là kiến trúc mạng theo kiểu feed-forward network, các lớp neurals sẽ là các fully-connected layers được xếp liên tiếp nhau. Đối với mạng DNN, Keras cung cấp 1 kiểu layers có tên là Dense (hay fully-connected-layers), cho phép chúng ta có thể xây dựng 1 Dense layer với các tham số cơ bản như: số neural trong layer đó, activation function (hàm kích hoạt), các khởi tạo tham số cho lần chạy đầu tiên và còn nhiều các tham số khác nữa. Trong 1 mạng DNN, ta có thể có nhiều lớp fully-connected layers, trong mỗi layer, các tham số như số neural, hàm kích hoạt có thể khác nhau, các bạn có thể thoải mái cài đặt theo ý thích.
2) CNN layers: CNN là một kiến trúc mạng neural rất thích hợp cho các bài toán

mà dữ liệu là ảnh hoặc video. Có 2 loại layer chính trong CNN: convolutional layer và pooling layer. Convolutional layer: 2 loại convolutional layer cơ bản đó là convolutional layer 1D và 2D.

- Convolutional layer 1D hoạt động với dữ liệu đầu vào là một ma trận 3 chiều.
- Convolutional layer 2D làm việc với dữ liệu đầu vào là ma trận 4 chiều.

Pooling layer: Keras hiện tại đang hỗ trợ nhiều loại pooling layers khác nhau như: max pooling, average pooling, global average pooling,... Để sử dụng ta cũng làm tương tự như với convolutional layers.

3) RNN layers: RNN là một kiến trúc mạng neuron rất phù hợp với dữ liệu dạng sequence. Với RNN, output của thời điểm hiện tại không chỉ phụ thuộc vào đầu vào tại thời điểm đó mà còn phụ thuộc cả vào đầu vào tại các thời điểm trước. Trên thực tế, RNN đã được áp dụng và gặt hái được nhiều thành công trên nhiều lĩnh vực thực tế, đặc biệt là các bài toán về xử lý ngôn ngữ tự nhiên. Với kiến trúc mạng RNN, có 2 loại layer thường được dùng bao gồm: GRU, LSTM. Bên cạnh đó Keras còn cung cấp wrapper cho phép ta sử dụng các layers RNN 2 chiều (bi-directional).

Mô tả thuật toán 1.

Thuật toán 1 $[G, Q] = Training(\mathcal{V}, R_{N\times C}, L)$: Tên thuật toán

Đầu vào:

1: $\mathcal{V}$: Input số 1;
2: $R_{N\times C}$: Input số 1;

Đầu ra:

3: $G = g_1, .., g_L$: Output số 1;
4: Ma trận Q: Output số 2 ;
5: **Bước 1:** Nội dung bước 1.
6: **Bước 2:** Nội dung bước 2.

4

Thực nghiệm

4.1 Tập dữ liệu

Dataset: Lấy từ project Fruits-360: A dataset of images containing fruits and vegetables. Version: 2019.08.14 (Ngày tải về 20/9/2019)
Link tải data: https://www.kaggle.com/moltean/fruits/download
Mô tả dataset:
Tổng số ảnh: 81120 ảnh
Tập training: 60498 ảnh (một trái cây hoặc rau củ trên 1 hình)
Tập test: 20622 ảnh (một trái cây hoặc rau củ trên 1 hình)
Số hình có nhiều loại trái cây hoặc rau củ: 103 hình.
Tổng số loại trái cây và rau củ: 120 (tương ứng 120 thư mục trong tập trainin, thư mục được đặt tên theo tên trái cây hoặc rau củ)
Kích thước ảnh : 100x100 pixels.
Định dạng tên ảnh: index_100.jpg (e.g. 32_100.jpg) hoặc r_image_index_100.jpg (e.g. r_32_100.jpg) hoặc r2_image_index_100.jpg hoặc r3_image_index_100.jpg. ”r” thể hiện trục xoay của trái cây khi chụp, ví dụ: ”r2” có nghĩa là trái cây đó đang xoay quanh trục thứ 3. 100 thể hiện cho kích thước ảnh (100x100 pixels).
Các giống khác nhau của cùng một loại trái cây (ví dụ như táo có táo đỏ, táo vàng,..) được lưu trữ ở các thư mục khác nhau.

4.2 Bố trí dữ liệu

4.3 Thiết lập thực nghiệm

Cấu hình máy 1: Win 10 Home 64 bit
Inspiron 13-5378 Signature Edition.
Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 Ghz.

RAM: 8.00 GB.
Cấu hình máy 2: Win 10 Pro 64 bit
Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz (4 CPUs).
RAM: 8.00 GB.
Cấu hình máy 3:
Intel(R) Xeon(R) CPU @ 2.20GHz 8 nhân.
RAM: 30 GB.

4.3.1 KNN và SVM

Chuẩn bị những thư viện cần thiết

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
import glob
import os
import mahotas
import h5py
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from os import listdir
from os.path import isfile, join, isdir
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder

```

Tạo đường dẫn đến các thư mục chứa dateset và đường dẫn đến thư mục output.

```
# define link
h5_dataTrain   = 'output/outputtrain/datatrain2.h5'
h5_labelsTrain = 'output/outputtrain/labelstrain2.h5'
h5_dataTest    = 'output/outputtest/datatest2.h5'
h5_labelsTest  = 'output/outputtest/labelstest2.h5'
```

Xây dựng các hàm trích xuất vector đặc trưng từ ảnh: HuMoment, Color Histogram, Heralick

```python

# Hàm trích xuất đặc trưng: Hu_moments, Haralick, Histogram
bins = 8
# feature-descriptor-1: Hu Moments
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature

# feature-descriptor-2: Haralick Texture
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # compute the haralick texture feature vector
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    # return the result
    return haralick

# feature-descriptor-3: Color Histogram
def fd_histogram(image, mask=None):
    # convert the image to HSV color-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    # compute the color histogram
    hist  = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
```

Tạo các mảng trống để lưu các vector đặc trưng của dataset trong quá trình trích xuất đặc trưng từ ảnh

```
51
52  # Tạo mảng rỗng lưu feature và label của 2 file train & test
53  featuresTrain = []
54  labelsTrain   = []
55  featuresTest  = []
56  labelsTest    = []
57
```

Đọc ảnh từ tập training trong dataset, trích xuất vector đặc trưng của ảnh và nhãn của nó.

```
# Lấy feature & label của image trong train
train_path = 'dataset/train'
my_path_list = [f for f in listdir(train_path) if isdir(train_path)]
train_labels = os.listdir(train_path)
train_labels.sort()

for my_path in my_path_list:
    onlyfiles = [f for f in listdir(join(train_path, my_path)) if isfile(join(train_path, my_path, f))]
    for file in onlyfiles:
        current_label = my_path
        image = cv2.imread(join(train_path, my_path, file))
        #fv_hu_moments = fd_hu_moments(image)
        fv_haralick   = fd_haralick(image)
        #fv_histogram  = fd_histogram(image)
        featureTrain = np.hstack([fv_haralick])
        # Cập nhật featureTrain và labelsTrain
        featuresTrain.append(featureTrain)
        labelsTrain.append(current_label)

    print('[Train feature]: {}'.format(current_label))
print('[Train feature]: Finish!')

```

Đọc ảnh từ tập test trong dataset, trích xuất vector đặc trưng của ảnh và nhãn của nó

```
79
80  # Lấy feature & label của image trong test
81  test_path = 'dataset/test'
82  my_path_list = [f for f in listdir(test_path) if isdir(test_path)]
83  test_labels = os.listdir(test_path)
84  test_labels.sort()
85
86  for my_path in my_path_list:
87      onlyfiles = [f for f in listdir(join(test_path, my_path)) if isfile(join(test_path, my_path, f))]
88      for file in onlyfiles:
89          current_label = my_path
90          image = cv2.imread(join(test_path, my_path, file))
91          #fv_hu_moments = fd_hu_moments(image)
92          fv_haralick   = fd_haralick(image)
93          #fv_histogram  = fd_histogram(image)
94          featureTest = np.hstack([fv_haralick])
95          # Cập nhật featureTrain và labelsTrain
96          featuresTest.append(featureTest)
97          labelsTest.append(current_label)
98      print('[Test feature]: {}'.format(current_label))
99  print('[Test feature]: Finish!')
100
```

Mã hóa các nhãn của ảnh thành số

```
100
101  # Mã hóa dữ liệu labels
102  le          = LabelEncoder()
103  targetTrain = le.fit_transform(labelsTrain)
104  targerTest  = le.fit_transform(labelsTest)
105  print('[Encoding]: Finish!')
106
```

lưu lại các vector đặc trưng của ảnh trong tập training và tập test cùng với nhãn tương ứng của chúng (sau khi đã mã hóa) trong các file.h5

```
106
107  # Lưu lại feature & label của train set vào file H5
108  h5f_dataTrain = h5py.File(h5_dataTrain, 'w')
109  h5f_dataTrain.create_dataset('dataset_1', data=np.array(featuresTrain))
110
111  h5f_labelTrain = h5py.File(h5_labelsTrain, 'w')
112  h5f_labelTrain.create_dataset('dataset_1', data=np.array(targetTrain))
113
114  h5f_dataTrain.close()
115  h5f_labelTrain.close()
116
117  # Lưu lại feature & label của test set vào file H5
118  h5f_dataTest = h5py.File(h5_dataTest, 'w')
119  h5f_dataTest.create_dataset('dataset_1', data=np.array(featuresTest))
120
121  h5f_labelTest = h5py.File(h5_labelsTest, 'w')
122  h5f_labelTest.create_dataset('dataset_1', data=np.array(targerTest))
123
124  h5f_dataTest.close()
125  h5f_labelTest.close()
126
127  print('[SAVING] Finish!')
128
```

Đọc các file.h5 đã lưu

```
# Đọc dữ liệu từ file H5
h5f_dataTrain  = h5py.File(h5_dataTrain, 'r')
h5f_labelTrain = h5py.File(h5_labelsTrain, 'r')

global_features_string1 = h5f_dataTrain['dataset_1']
global_labels_string1   = h5f_labelTrain['dataset_1']

training_features = np.array(global_features_string1)
training_labels   = np.array(global_labels_string1)

h5f_dataTrain.close()
h5f_labelTrain.close()

h5f_dataTest  = h5py.File(h5_dataTest, 'r')
h5f_labelTest = h5py.File(h5_labelsTest, 'r')

global_features_string2 = h5f_dataTest['dataset_1']
global_labels_string2   = h5f_labelTest['dataset_1']

test_features = np.array(global_features_string2)
test_labels   = np.array(global_labels_string2)

h5f_dataTest.close()
h5f_labelTest.close()
print('[READING] Finish!')

```

Huấn luyện với KNN

```python
# Mô hình KNN
print('Begin Training with KNN ....')
knn_clf = KNeighborsClassifier(n_jobs=-1, weights='distance', n_neighbors=2)
knn_clf.fit(training_features, training_labels)
y_knn_pred = knn_clf.predict(test_features)
print('End of Training....')
print('KNN accuracy score: ',accuracy_score(test_labels, y_knn_pred)*100,'%')
```

Huấn luyện với SVM

```
# Mô hình SVM
print('Begin Training with SVM ....')
svm_clf = SVC(gamma='auto',probability = True)
svm_clf.fit(training_features, training_labels)
y_svm_pred = svm_clf.predict(test_features)
print('End of Training....')
print('SVM accuracy score: ',accuracy_score(test_labels, y_svm_pred)*100,'%')
```

4.3.2 Convolutional neural network (CNN)

```
from sklearn.datasets import load_files
import numpy as np
from keras.preprocessing.image import array_to_img, img_to_array,
    load_img
import matplotlib.pyplot as plt
import cv2

train_dir = '360/Training'
test_dir = '360/Test'

def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets = np.array(data['target'])
    target_labels = np.array(data['target_names'])
    return files,targets,target_labels

x_train, y_train,target_labels = load_dataset(train_dir)
x_test, y_test,_ = load_dataset(test_dir)
print('Loading complete!')

print('Training set size : ' , x_train.shape[0])
print('Testing set size : ', x_test.shape[0])

no_of_classes = len(np.unique(y_train))
print(no_of_classes)



from keras.utils import np_utils
y_train = np_utils.to_categorical(y_train,no_of_classes)
y_test = np_utils.to_categorical(y_test,no_of_classes)

x_test,x_valid = x_test[9000:],x_test[:9000]
y_test,y_vaild = y_test[9000:],y_test[:9000]
print('Vaildation X : ',x_valid.shape)
```

```
print('Vaildation y :',y_vaild.shape)
print('Test X : ',x_test.shape)
print('Test y : ',y_test.shape)

def convert_image_to_array(files):
    images_as_array=[]
    for file in files:
        # Convert to Numpy Array
        images_as_array.append(img_to_array(load_img(file)))
    return images_as_array

x_train = np.array(convert_image_to_array(x_train))
print('Training set shape : ',x_train.shape)

x_valid = np.array(convert_image_to_array(x_valid))
print('Validation set shape : ',x_valid.shape)

x_test = np.array(convert_image_to_array(x_test))
print('Test set shape : ',x_test.shape)

print('1st training image shape ',x_train[0].shape)


# time to re-scale so that all the pixel values lie within 0 to 1
#x_train = x_train.astype('float32')/255
#x_valid = x_valid.astype('float32')/255
#x_test = x_test.astype('float32')/255


#Let's visualize the first 10 training images!

#Hien 10 hinh len để xem
# fig = plt.figure(figsize =(30,5))
# for i in range(10):
#     ax = fig.add_subplot(2,5,i+1,xticks=[],yticks=[])
#     ax.imshow(np.squeeze(x_train[i]))
#plt.show()
```

```
73
74 #Simple CNN from scratch - we are using 3 Conv layers followed by
       maxpooling layers.
75 # At the end we add dropout, flatten and some fully connected
     layers(Dense).
76 print('Building a model...')
77 from keras.models import Sequential
78 from keras.layers import Conv2D,MaxPooling2D
79 from keras.layers import Activation, Dense, Flatten, Dropout
80 from keras.preprocessing.image import ImageDataGenerator
81 from keras.callbacks import ModelCheckpoint
82 from keras import backend as K
83
84 model = Sequential()
85 model.add(Conv2D(filters = 16, kernel_size = 2,input_shape
     =(100,100,3),padding='same'))
86 model.add(Activation('relu'))
87 model.add(MaxPooling2D(pool_size=2))
88
89 model.add(Conv2D(filters = 32,kernel_size = 2,activation= 'relu',
     padding='same'))
90 model.add(MaxPooling2D(pool_size=2))
91
92 model.add(Conv2D(filters = 64,kernel_size = 2,activation= 'relu',
     padding='same'))
93 model.add(MaxPooling2D(pool_size=2))
94
95 model.add(Conv2D(filters = 128,kernel_size = 2,activation= 'relu'
     ,padding='same'))
96 model.add(MaxPooling2D(pool_size=2))
97
98 model.add(Dropout(0.3))
99 model.add(Flatten())
100 model.add(Dense(150))
101 model.add(Activation('relu'))
102 model.add(Dropout(0.4))
103 model.add(Dense(120,activation = 'softmax'))
```

```
#model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
print('Compiled init Model!')



print('Start to traing...')
batch_size = 32
checkpointer = ModelCheckpoint(filepath = '
   cnn_from_scratch_fruits.hdf5', verbose = 1, save_best_only =
   True)
history = model.fit(x_train,y_train,
        batch_size = 32,
        epochs=30,
        validation_data=(x_valid, y_vaild),
        callbacks = [checkpointer],
        verbose=2, shuffle=True)

print('End of traing...')
# model.save('cnnS2.h5')

# load the weights that yielded the best validation accuracy
model.load_weights('cnn_from_scratch_fruits.hdf5')
score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])



# y_pred = model.predict(x_test)
# # plot a random sample of test images, their predicted labels,
   and ground truth
# fig = plt.figure(figsize=(16, 9))
# for i, idx in enumerate(np.random.choice(x_test.shape[0], size
   =16, replace=False)):
```

```
#     ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
#     ax.imshow(np.squeeze(x_test[idx]))
#     pred_idx = np.argmax(y_pred[idx])
#     true_idx = np.argmax(y_test[idx])
#     ax.set_title("{} ({})".format(target_labels[pred_idx],
    target_labels[true_idx]),
#                 color=("green" if pred_idx == true_idx else "
    red"))
# plt.show()


#Finally lets visualize the loss and accuracy wrt epochs

# import matplotlib.pyplot as plt
# plt.figure(1)

#  # summarize history for accuracy

# plt.subplot(211)
# plt.plot(history.history['acc'])
# plt.plot(history.history['val_acc'])
# plt.title('model accuracy')
# plt.ylabel('accuracy')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')

#  # summarize history for loss

# plt.subplot(212)
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])
# plt.title('model loss')
# plt.ylabel('loss')
# plt.xlabel('epoch')
# plt.legend(['train', 'test'], loc='upper left')
# plt.show()
```

4.4 Kết quả thực nghiệm

K-Nearest	5	10	50	100	500
Feature Vector					
Hu Moments Haralick Histogram	98.29	98.13	96.28	94.57	88.71
Hu Moments Haralick	81.37	80.66	75.26	71.87	62.76
Hu Moments Histogram	98.12	97.92	95.94	93.81	87.61
Haralick Histogram	98.25	98.10	96.43	94.85	89.72
Hu Moments	10.17	11.05	12.85	13.45	16.18
Haralick	79.73	78.99	74.44	71.86	65.51
Histogram	98.10	97.91	96.10	94.07	88.56
HOG	82.84	80.90	70.73	64.34	49.58

Table 1: Bảng số liệu về kết quả thực nghiệm của KNN (đơn vị %)

K-Nearest	3	11	51	101	501
Feature Vector					
Hu Moments Haralick Histogram	98.43	97.98	96.25	94.56	88.69
Hu Moments Haralick	81.77	80.18	75.25	71.82	62.72
Hu Moments Histogram	98.28	97.81	95.91	93.78	87.59
Haralick Histogram	98.40	97.96	96.35	94.82	89.72
Hu Moments	9.69	11.23	12.86	13.44	16.21
Haralick	80.19	78.72	74.34	71.82	65.49
Histogram	98.27	97.80	96.06	94.06	88.54
HOG	84.62	80.70	69.62	62.40	43.06

Table 2: Bảng số liệu về kết quả thực nghiệm của KNN (đơn vị %)

SVM	Kernel = Rbf	Kernel = Linear
Feature Vector		
Hu Moments Haralick Histogram	73.34	98.87
Hu Moments Haralick	58.40	71.30
Hu Moments Histogram	63.79	98.55
Haralick Histogram	73.38	98.85
Hu Moments	8.25	11.37
Haralick	61.73	69.75
Histogram	63.69	98.52
HOG	89.25	87.74

Table 3: Bảng số liệu về kết quả thực nghiệm của SVM (đơn vị %)

4.5 Thảo luận

Kết quả thu được:
Đối với cách trích xuất đặc trưng cảu trái cây theo phương pháp Hu Moments, độ chính xác ở cả 2 phương pháp học KNN và SVM đều cho kết quả khá thấp. Kết quả này là do với phương pháp trích xuất đặc trưng theo Hu Moments, vector đặc trưng thu được thể hiện hình dạng của trái cây. Do đó với những trái cây có cùng hình dáng hoặc những trái cây trong cùng 1 lớp nhưng khác góc chụp thì với phương pháp Hu Moments không thể phân biệt chính xác được

Đối với phương pháp trích xuất đặc trưng theo Haralick Texture, kết quả thu được tương đối cao. Heralick vẫn còn có sự nhầm lẫn giữa 2 loại trái cây có họa tiết hoa văn tương đối giống nhau. Ví Dụ: Pineapple và Rambutan

Đối với phương pháp trích xuất đặc trưng theo Color Histogram thì kết quả thu được cao. Nguyên nhân đối với bộ dataset trái cây, ở tập training là các ảnh chụp bởi cùng 1 quả, do đó khả năng các trái cây có cùng color histogram hoặc có color histogram tương đối giống nhau là rất thấp.

Đối với phương pháp trích xuất đặc trưng theoo Histogram of oriented gradient kết quả thu được ở mức tương đối cao.

Đối với phương pháp học KNN khi ta tăng số K thì độ chính xác sẽ bị giảm dần.
Đối với phương pháp học SVM thì kết quả thu được khi ta sử dụng Kernel là Linear thường cao hơn Kernel là RBF.
Đối với Mạng tích chập (CNN) kết quả thu được rất cao.

5

Kết luận và Hướng phát triển

Bài toán nhận dạng trái cây được thực hiện thông qua các phương pháp trích xuất đặc trưng như: Haralick, Hu moment, Color Histogram, HisTogram of oriented gradient. Các phương pháp học áp dụng là KNN và SVM
Đối với bộ dataset sử dụng (Trái cây) đạt được độ chính xác cao với CNN (98.13 %) ngoài ra khi sử dụng kĩ thuật trích xuất đặc trưng ảnh là Color Histogram hoặc Histogram of Oriented gradient thì với cả 2 phương pháp học KNN và SVM đều cho kết quả khá cao.
Đối với phương pháp học máy KNN đối với bộ dữ liệu này khi chạy thực nghiệm, nếu ta chọn k càng lớn thì độ chính sác dự đoán của mô hình cảng giảm. việc chọn k là số chẵn hoặc số lẻ ví dụ chọn k = 10 hoặc k = 11 thì độ chính sác không bị chênh lệch quá nhiều.
Đối với phương pháp học máy SVM, đối với bộ dữ liệu này, khi huấn luyện mô hình với Kernel sử dụng là Linear thì độ chính sác thu được thường cao hơn khi huấn luyện mô hình với kernel sử dụng là RBF.

Hướng phát triển tiếp theo: Tiếp cận và cài đặt với mô hình mạng học sâu VGG16 và VGG19, đây có thể coi là những mô hình học hiệu quả nhất với bộ dữ liệu trái cây [7].

References

[1] https://www.kaggle.com/moltean/fruits

[2] S.Arivazhagan, R.Newlin Shebiah, S.Selva Nidhyanandhan, L.Ganesan *Fruit Recognition using Color and Texture Features*
Link:https://pdfs.semanticscholar.org/7537/2f88692c8a7de53b024ed9f5be81b5f002f1.pdf?_g
2050126797.1570848986

[3] Gogul Ilango, *Image Classification using Python and Scikit-learn*.
Link:https://gogul.dev/software/image-classification-python?fbclid=IwAR0-Y4OfPrwG3aAqDeNhPDQCIZWgRQA-w7ul1sawcpPKh4uy8GfMwYV5OG0

[4] Nadya Aditama, *Computer Vision: Fruit Recognition*.
Link:https://medium.com/datadriveninvestor/computer-vision-fruit-recognition-4f2f7a780e00

[5] Mihai Oltean and Horea Muresan *Fruit recognition from images using deep learning*
Linh:https://github.com/Horea94/Fruit-Images-Dataset/tree/master/papers

[6] Riad Shaltaf *Fruit image recognition*
link:https://www.kaggle.com/shaltaf/fruit-image-recognition

[7] Anindita Pani *CNN from scratch with 98% accuracy*
Link:https://www.kaggle.com/aninditapani/cnn-from-scratch-with-98-accuracy

[8] IS *Fruits-360 - Transfer Learning using Keras*
Link:https://www.kaggle.com/amadeus1996/fruits-360-transfer-learning-using-keras

[9] Horea Muresan, *Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.*

[10] Bế Ngọc Trọng, *Nhận dạng ảnh cơ bản với Python*. Link:https://viblo.asia/p/nhan-dang-anh-co-ban-voi-python-bWrZn6mbZxw_ket-luan-10g

[11] Horea Muresan, *Fruit-360 CNN TensorFlown*. Link:https://www.kaggle.com/mitch9090/fruit-360-cnn-tensorflow

[12] Shivam Bansal. *CNN Architectures : VGG, ResNet, Inception + TL* Link:https://www.kaggle.com/shivamb/cnn-architectures-vgg-resnet-inception-tl

[13] TopDev. *Thuật toán CNN – Convolutional Neural Network* Link:https://topdev.vn/blog/thuat-toan-cnn-convolutional-neural-network/

[14] Nguyen The Toan. *Sử dụng CNN trong bài toán nhận dạng mặt người* Link:https://viblo.asia/p/su-dung-cnn-trong-bai-toan-nhan-dang-mat-nguoi-phan-1-eW65GoOP5DO

[15] Nguyễn Phúc Lương. *Ứng dụng Convolutional Neural Network trong bài toán phân loại ảnh* Link:https:https://viblo.asia/p/ung-dung-convolutional-neural-network-trong-bai-toan-phan-loai-anh-4dbZNg8ylYM

[16] Học Cùng Nhau. *Mạng nơron tích chập* Link:http://cuoichovui.com/python/mang-noron-tich-chap/

[17] Nguyễn Phương Lan. *Tìm hiểu về hog(histogram of oriented gradients)* Link:http://cuoichovui.com/python/mang-noron-tich-chap/

[18] Hải Hà. *Tìm hiểu về hog(histogram of oriented gradients)* Link:https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-of-oriented-gradients-V3m5WAwxZO7

[19] Hoang Dinh thoi. *Deep Learning quá khó? Đừng lo, đã có Keras.*
Link:https://viblo.asia/p/deep-learning-qua-kho-dung-lo-da-co-keras-LzD5dBqoZjY

[20] Hải Hà. *Tìm hiểu về hog(histogram of oriented gradients)*
Link:https://viblo.asia/p/tim-hieu-ve-phuong-phap-mo-ta-dac-trung-hog-histogram-of-oriented-gradients-V3m5WAwxZO7

[21] Gogul. *Texture Recognition using Haralick Texture and Python*
Link:https://gogul.dev/software/texture-recognition

[22] Ntppro. *[Image Descriptor] Haralick Texture Features*
Link:https://codelungtung.wordpress.com/2018/07/13/image-descriptor-haralick-texture/

[23] Robert M.haralick, K. Shanmungam, Its'Hak Dinstein. *[Texture Features for Image Classification*

[24] Lou Marvin Caraig. *Understanding image histograms with OpenCV*
link: https://lmcaraig.com/understanding-image-histograms-with-opencv

[25] Adrian Rosebrock . *Clever Girl: A Guide to Utilizing Color Histograms for Computer Vision and Image Search Engines*
Link: https://www.pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines

[26] *Image Histograms*
Link:https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20162017/LectureNotes/IP/Images/Ima

[27] Adrian Rosebrock. *OpenCV Shape Descriptor: Hu Moments Example*
Link: https://www.pyimagesearch.com/2014/10/27/opencv-shape-descriptor-hu-moments-example

[28] Satya Mallick. *Shape Matching using Hu Moments (C++/Python)*
Link: https://www.learnopencv.com/shape-matching-using-hu-moments-c-python

[29] Satya Mallick. *Histogram of Oriented Gradients*
Link: https://www.learnopencv.com/histogram-of-oriented-gradients/