ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**🙠🟑🙢**

**ĐỒ ÁN CUỐI KHÓA MÔN HỌC**

**NHẬN DẠNG THỊ GIÁC VÀ ỨNG DỤNG**

Đề tài:

**NHẬN DẠNG CHỮ VIẾT TAY**

**GVHD:** TS. Lê Đình Duy

TS. Nguyễn Tấn Trần Minh Khang

**HVTH:** Trần Ngọc Đông

**MÃ SỐ:** CH1601002

2017

Mục lục

[I. Dữ liệu 3](#_Toc488057011)

[II. Chương trình 6](#_Toc488057012)

[2.1. Đọc dữ liệu từ tập huấn luyện và tập test 6](#_Toc488057013)

[2.2. Nhận dạng chữ viết với đặc trưng RAW 6](#_Toc488057014)

[2.2.1. Sử dụng bộ phân lớp KNN 6](#_Toc488057015)

[2.2.2. Sử dụng bộ phân lớp SVM 7](#_Toc488057016)

[2.3. Nhận dạng chữ viết với đặc trưng HOG 8](#_Toc488057017)

[2.3.1. Sử dụng bộ phân lớp KNN 9](#_Toc488057018)

[2.3.2. Sử dụng bộ phân lớp SVM 10](#_Toc488057019)

[2.4. Nhận dạng chữ viết với đặc trưng LBP 11](#_Toc488057020)

[2.4.1. Sử dụng bộ phân lớp KNN 11](#_Toc488057021)

[2.4.2. Sử dụng bộ phân lớp SVM 13](#_Toc488057022)

[2.5. Nhận dạng chữ viết với mô hình Bag of Words (BoW) 15](#_Toc488057023)

[2.6. Nhận dạng chữ viết với Deep learning 16](#_Toc488057024)

[2.7. Giao diện chương trình 19](#_Toc488057025)

[III. Tìm hiểu thuật toán Support Vector Machine (SVM) 20](#_Toc488057026)

[3.1. Phân hai lớp với SVM: 21](#_Toc488057027)

[3.2. Phân nhiều lớp với SVM 22](#_Toc488057028)

[3.3. Các bước chính của SVM 22](#_Toc488057029)

[TÀI LIỆU THAM KHẢO 24](#_Toc488057030)

# **I. Dữ liệu**

- Tập huấn luyện:

• Training set images: gồm 60000 ảnh

Link download: http://yann.lecun.com/exdb/mnist/train-images-idx3-  
ubyte.gz

• Training set labels

Link download: http://yann.lecun.com/exdb/mnist/train-labels-idx1-  
ubyte.gz

- Tập test:

• Test set images: gồm 10000 ảnh

Link download: http://yann.lecun.com/exdb/mnist/t10k-images-idx3-  
ubyte.gz

• Test set labels

Link download: http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-  
ubyte.gz

- Code xây dựng Data để dùng cho mô hình Bag of Words và Deep learning.

function builDataTrain( )

%% Load Data Train

strData = 'train-images.idx3-ubyte';

strDataLabel = 'train-labels.idx1-ubyte';

[imgDataTrain,lblDataTrain] = loadData(strData,strDataLabel);

%% Create Folder Data

mkdir DataTrain;

mkdir DataTrain 0;

mkdir DataTrain 1;

mkdir DataTrain 2;

mkdir DataTrain 3;

mkdir DataTrain 4;

mkdir DataTrain 5;

mkdir DataTrain 6;

mkdir DataTrain 7;

mkdir DataTrain 8;

mkdir DataTrain 9;

%% Build Data Train

nCount = zeros(1,10);

nDataTrain = size(imgDataTrain,2);

for i = 1:nDataTrain

img1D = imgDataTrain(:,i);

img2D = reshape(img1D,28,28);

lblI = lblDataTrain(i);

nCount(lblI+1) = nCount(lblI+1) + 1;

nID = nCount(lblI+1);

if(nID<10)

strNumber = ['000' num2str(nID)];

elseif(nID<100)

strNumber = ['00' num2str(nID)];

elseif(nID<1000)

strNumber = ['0' num2str(nID)];

else

strNumber = num2str(nID);

end

strNameFile = ['DataTrain\' num2str(lblI) '\image\_' strNumber '.jpg'];

imwrite(img2D,strNameFile);

end

end

===============================================

function buildDataTest( )

%% Load Data Test

strData = 't10k-images.idx3-ubyte';

strDataLabel = 't10k-labels.idx1-ubyte';

[imgDataTest,lblActualDataTest] = loadData(strData,strDataLabel);

%% Create Folder Data

mkdir Test;

mkdir Test 0;

mkdir Test 1;

mkdir Test 2;

mkdir Test 3;

mkdir Test 4;

mkdir Test 5;

mkdir Test 6;

mkdir Test 7;

mkdir Test 8;

mkdir Test 9;

%% Build Data Test

nCount = zeros(1,10);

nDataTest = size(imgDataTest,2);

for i = 1:nDataTest

img1D = imgDataTest(:,i);

img2D = reshape(img1D,28,28);

lblI = lblActualDataTest(i);

fprintf('\n lblActualDataTest(i): %d',lblI);

fprintf('\n nCount(lblI+1): %d',nCount(lblI+1));

nCount(lblI+1) = nCount(lblI+1) + 1;

fprintf('\n nCount(lblI+1) + 1: %d',nCount(lblI+1)+1);

nID = nCount(lblI+1);

fprintf('\n nID: %d',nID);

if(nID<10)

strNumber = ['000' num2str(nID)];

elseif(nID<100)

strNumber = ['00' num2str(nID)];

elseif(nID<1000)

strNumber = ['0' num2str(nID)];

else

strNumber = num2str(nID);

end

strNameFile = ['Test\' num2str(lblI) '\image\_' strNumber '.jpg'];

imwrite(img2D,strNameFile);

end

end

# **II. Chương trình**

## **2.1. Đọc dữ liệu từ tập huấn luyện và tập test**

- Sử dụng hai lệnh: loadMNISTImages và loadMNISTLabels được phát triển bởi Standford.

Link download: http://ufldl.stanford.edu/wiki/  
resources/mnistHelper.zip

## **2.2. Nhận dạng chữ viết với đặc trưng RAW**

Kỹ thuật nhận dạng chữ đơn giản nhất dựa trên cơ sở đối sánh các nguyên mẫu (prototype) với nhau để nhận dạng kí tự hoặc từ. Nói chung, toán tử đối sánh xác định mức độ giống nhau giữa hai vector (nhóm các điểm, hình dạng, độ cong, …) trong một không gian đặc trưng.

### ***2.2.1. Sử dụng bộ phân lớp KNN***

function nImgTrue = raw( )

lblTestLabelsAll = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist('RAWKNN.mat', 'file') == 2

load('RAWKNN.mat');

else

imgTrainImagesAll = loadMNISTImages('train-images.idx3-ubyte');

lblTrainLabelsAll = loadMNISTLabels('train-labels.idx1-ubyte');

Md1 = fitcknn(imgTrainImagesAll', lblTrainLabelsAll);

imgTestImagesAll = loadMNISTImages('t10k-images.idx3-ubyte');

lblResult = predict(Md1,imgTestImagesAll');

save('RAWKNN.mat','lblResult');

end

nResult = (lblResult == lblTestLabelsAll);

nImgTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh là 9691, chiếm 96.91%.

### ***2.2.2. Sử dụng bộ phân lớp SVM***

function nImgTrue = rawSvm( )

lblTestLabelsAll = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist('RAWSVM.mat', 'file') == 2

load('RAWSVM.mat');

else

imgTrainImagesAll = loadMNISTImages('train-images.idx3-ubyte');

lblTrainLabelsAll = loadMNISTLabels('train-labels.idx1-ubyte');

Md1 = fitcecoc(imgTrainImagesAll', lblTrainLabelsAll);

imgTestImagesAll = loadMNISTImages('t10k-images.idx3-ubyte');

lblResult = predict(Md1,imgTestImagesAll');

save('RAWSVM.mat','lblResult');

end

nResult = (lblResult == lblTestLabelsAll);

nImgTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh là 9438, chiếm 94.38%.

## **2.3. Nhận dạng chữ viết với đặc trưng HOG**

Các đặc trưng HOG ban đầu được thiết kế để phát hiện người (human detection) trong dữ liệu ảnh nhưng sau đó được mở rộng và được sử dụng rộng rãi cho các bài toán phát hiện đối tượng nối chung (object detection). Phương pháp HOG dựa trên việc đếm số lần xuất hiện cảu các hướng đạo hàm (gradient orientation) trong các vùng cục bộ của ảnh.

Bản chất của phương pháp HOG là các thông tin về hình dáng và vẻ bề ngoài của các đối tượng cục bộ trong ảnh có thể được mô tả bằng cách sử dụng thông tin về sự phân bố của các cường độ gradient (intensity gradients) hoặc của các hướng biên (edge directions). Các toán tử HOG được cài đặt bằng cách chia nhỏ các bức ảnh thành các vùng con được gọi là các “tế bào” (cell) và với mỗi cell đó một histogram về các hướng của gradients sẽ được tính cho các điểm nằm trong cell. Ghép các histogram này lại ta sẽ có một biểu diễn cho bức ảnh ban đầu. Để tăng cường hiệu năng nhận dạng, các histogram cục bộ có thể được chuẩn hóa về độ tương phản bằng cách tính một ngưỡng cường độ trong một vùng lớn hơn của ảnh, gọi là các khối (blocks) và sử dụng giá trị ngưỡng đó để chuẩn hóa tất cả các cell trong khối. Kết quả của bước chuẩn hóa này là vector đặc trưng sẽ có tính bất biến cao hơn đối với các thay đổi về điều kiện ánh sáng.

Các bước cơ bản của việc tính một vector HOG cho một ảnh gồm có:

• Tính gradient

• Gán hướng vào các vùng giá trị khác nhau

• Tính histogram của các khối

• Chuẩn hóa các khối

### ***2.3.1. Sử dụng bộ phân lớp KNN***

function nImageTrue = hog(m, n)

lblTest = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist(['HOG',num2str(m),num2str(n),'.mat'], 'file') == 2

load(['HOG',num2str(m),num2str(n),'.mat']);

else

imgTrain = loadMNISTImages('train-images.idx3-ubyte');

lblTrain = loadMNISTLabels('train-labels.idx1-ubyte');

imgI = imgTrain(:,1);

img2D = reshape(imgI,28,28);

[featureVector,hogVisualization] = extractHOGFeatures(img2D,'CellSize',[m m]);

nBins = size(featureVector,2);

nNumTrain = size(imgTrain, 2);

imgTrain\_hist = zeros(nBins, nNumTrain);

for i=1:nNumTrain

imgI = imgTrain(:,i);

img2D = reshape(imgI,28,28);

[featureVector,hogVisualization] = extractHOGFeatures(img2D,'CellSize',[m n]);

imgTrain\_hist(:,i) = featureVector;

end

Mdl = fitcknn(imgTrain\_hist',lblTrain);

imgTest = loadMNISTImages('t10k-images.idx3-ubyte');

nNumTest = size(imgTest,2);

imgTest\_hist = zeros(nBins, nNumTest);

for i=1:nNumTest

imgI = imgTest(:,i);

img2D = reshape(imgI,28,28);

imgTest\_hist(:,i) = extractHOGFeatures(img2D,'CellSize',[m n]);

end

lblResult = predict(Mdl,imgTest\_hist');

save(['HOG',num2str(m),num2str(n),'.mat'],'lblResult');

end

nResult = (lblResult == lblTest);

nImageTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh:

• CellSize [2 2] là: 9772 chiếm 97.72%

• CellSize [4 4] là: 9767 chiếm 97.67%

• CellSize [8 8] là: 9709 chiếm 97.09%

### ***2.3.2. Sử dụng bộ phân lớp SVM***

function nImageTrue = hogSvm(m,n)

lblTest = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist(['HOGSVM',num2str(m),num2str(n),'.mat'], 'file')== 2

load(['HOGSVM',num2str(m),num2str(n),'.mat']);

else

imgTrain = loadMNISTImages('train-images.idx3-ubyte');

lblTrain = loadMNISTLabels('train-labels.idx1-ubyte');

imgI = imgTrain(:,1);

img2D = reshape(imgI,28,28);

[featureVector,hogVisualization] = extractHOGFeatures(img2D,'CellSize',[m m]);

nBins = size(featureVector,2);

nNumTrain = size(imgTrain, 2);

imgTrain\_hist = zeros(nBins, nNumTrain);

for i=1:nNumTrain

imgI = imgTrain(:,i);

img2D = reshape(imgI,28,28);

[featureVector,hogVisualization] = extractHOGFeatures(img2D,'CellSize',[m n]);

imgTrain\_hist(:,i) = featureVector;

end

Mdl = fitcecoc(imgTrain\_hist',lblTrain);

imgTest = loadMNISTImages('t10k-images.idx3-ubyte');

nNumTest = size(imgTest,2);

imgTest\_hist = zeros(nBins, nNumTest);

for i=1:nNumTest

imgI = imgTest(:,i);

img2D = reshape(imgI,28,28);

imgTest\_hist(:,i) = extractHOGFeatures(img2D,'CellSize',[m n]);

end

lblResult = predict(Mdl,imgTest\_hist');

save(['HOGSVM',num2str(m),num2str(n),'.mat'],'lblResult');

end

nResult = (lblResult == lblTest);

nImageTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh:

• CellSize [4 4] là: 9883 chiếm 98.83%

• CellSize [8 8] là: 9794 chiếm 97.94%

## **2.4. Nhận dạng chữ viết với đặc trưng LBP**

LBP là viết tắt của Local Binary Pattern hay là mẫu nhị phân địa phương được Ojala trình bày vào năm 1996 như là một cách đo độ tương phản cục bộ của ảnh. Phiên bản đầu tiên của LBP được dùng với 8 điểm ảnh xung quanh và sử dụng giá trị của điểm ảnh ở trung tâm làm ngưỡng. Giá trị LBP được xác định bằng cách nhân các giá trị ngưỡng với trọng số ứng với mỗi điểm ảnh sau đó cộng tổng lại.

### ***2.4.1. Sử dụng bộ phân lớp KNN***

function nImgTrue = lbp(m,n)

lblTest = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist(['LBP',num2str(m),num2str(n),'.mat'], 'file') == 2

load(['LBP',num2str(m),num2str(n),'.mat']);

else

imgTrain = loadMNISTImages('train-images.idx3-ubyte');

lblTrain = loadMNISTLabels('train-labels.idx1-ubyte');

imgI = imgTrain(:,1);

img2D = reshape(imgI,28,28);

featureVector = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

nSize = length(featureVector);

nNumTrain = size(imgTrain, 2);

featuresDataTrain = zeros(nSize, nNumTrain);

for i=1:nNumTrain

imgI = imgTrain(:,i);

img2D = reshape(imgI,28,28);

featuresDataTrain(:,i) = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

end

Mdl = fitcknn(featuresDataTrain',lblTrain);

imgTest = loadMNISTImages('t10k-images.idx3-ubyte');

nNumTest = size(imgTest,2);

featuresDataTest = zeros(nSize, nNumTest);

for i=1:nNumTest

imgI = imgTest(:,i);

img2D = reshape(imgI,28,28);

featuresDataTest(:,i) = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

end

lblResult = predict(Mdl,featuresDataTest');

save(['LBP',num2str(m),num2str(n),'.mat'],'lblResult');

end

nResult = (lblResult == lblTest);

nImgTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh:

• NumNeighbors = 6 và Radius = 3 là: 70.79 chiếm 70.79%

• NumNeighbors = 8 và Radius = 2 là: 7137 chiếm 71.37%

• NumNeighbors = 8 và Radius = 3 là: 7320 chiếm 73.20%

• NumNeighbors = 8 và Radius = 4 là: 6652 chiếm 66.52%

• NumNeighbors = 10 và Radius = 3 là: 7041 chiếm 70.41%

• NumNeighbors = 24 và Radius = 5 là: 4851 chiếm 48.51%

### ***2.4.2. Sử dụng bộ phân lớp SVM***

function nImgTrue = lbpSvm(m,n)

lblTest = loadMNISTLabels('t10k-labels.idx1-ubyte');

if exist(['LBPSVM',num2str(m),num2str(n),'.mat'], 'file')== 2

load(['LBPSVM',num2str(m),num2str(n),'.mat']);

else

imgTrain = loadMNISTImages('train-images.idx3-ubyte');

lblTrain = loadMNISTLabels('train-labels.idx1-ubyte');

imgI = imgTrain(:,1);

img2D = reshape(imgI,28,28);

featureVector = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

nSize = length(featureVector);

nNumTrain = size(imgTrain, 2);

featuresDataTrain = zeros(nSize, nNumTrain);

for i=1:nNumTrain

imgI = imgTrain(:,i);

img2D = reshape(imgI,28,28);

featuresDataTrain(:,i) = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

end

Mdl = fitcecoc(featuresDataTrain',lblTrain);

imgTest = loadMNISTImages('t10k-images.idx3-ubyte');

nNumTest = size(imgTest,2);

featuresDataTest = zeros(nSize, nNumTest);

for i=1:nNumTest

imgI = imgTest(:,i);

img2D = reshape(imgI,28,28);

featuresDataTest(:,i) = extractLBPFeatures(img2D,'NumNeighbors',m,'Radius',n);

end

lblResult = predict(Mdl,featuresDataTest');

save(['LBPSVM',num2str(m),num2str(n),'.mat'],'lblResult');

end

nResult = (lblResult == lblTest);

nImgTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh:

• NumNeighbors = 6 và Radius = 3 là: 7079 chiếm 70.79%

• NumNeighbors = 8 và Radius = 1 là: 5389 chiếm 53.89%

• NumNeighbors = 8 và Radius = 3 là: 7059 chiếm 70.58%

• NumNeighbors = 8 và Radius = 4 là: 6743 chiếm 67.43%

• NumNeighbors = 8 và Radius = 5 là: 6555 chiếm 65.55%

• NumNeighbors = 10 và Radius = 3 là: 7041 chiếm 7041%

## **2.5. Nhận dạng chữ viết với mô hình Bag of Words (BoW)**

Bag of words là một phương pháp để biểu diễn ảnh nhằm giải quyết vấn đề không gian lưu trữ và tốc độ truy vấn bằng phương pháp phân cụm. Trình tự các bước thực hiện của phương pháp này, như sau:

- Phân cụm bằng giải thuật k-means.

- Gán từng descriptor của ảnh vào cluster gần nhất.

- Đếm số descriptor của ảnh được gán vào từng cluster.

- Cuối cùng biểu diễn ảnh bằng 1 histogram các visual words.

BoW sử dụng giải thuật phân lớp là k-means với số lần lặp xác định để tạo ra k cluster bằng cách tính trung bình khoảng cách giữa các vectơ đặc trưng gần nhất. Tập hợp các cluster gọi là visual word vocabulary, kích thước của vocabulary phụ thuộc vào số lượng các cluster.

**Code**

function nImageTrue = bow( )

rootFolder = fullfile('DataTrain');

categories = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

imds = imageDatastore(fullfile(rootFolder, categories), 'LabelSource', 'foldernames');

tbl01 = countEachLabel(imds);

minSetCount = min(tbl01{:, 2});

imds = splitEachLabel(imds, minSetCount, 'randomize');

tbl02 = countEachLabel(imds);

if exist('BoW.mat', 'file') == 2 % Kiem tra neu ton tai thi doc len (Giup chuong trinh chay nhanh hon)

load('BoW.mat');

else

bag = bagOfFeatures(imds);

save('BoW.mat', 'bag');

end

img = readimage(imds, 1);

featureVector = encode(bag, img);

if exist('BoWClassifier.mat', 'file') == 2

load('BoWClassifier.mat');

else

categoryClassifier = trainImageCategoryClassifier(imds, bag);

save('BoWClassifier.mat', 'categoryClassifier');

end

rootFolder = fullfile('DataTest');

categories = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

imds = imageDatastore(fullfile(rootFolder, categories), 'LabelSource', 'foldernames');

tbl01 = countEachLabel(imds);

if exist('evaluateBoW.mat','file') == 2

load('evaluateBoW.mat');

else

confMatrixTest = evaluate(categoryClassifier, imds);

save('evaluateBoW.mat','confMatrixTest');

end

nImageTrue = mean(diag(confMatrixTest))\*100;

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh là 94.7989%.

## **2.6. Nhận dạng chữ viết với Deep learning**

Deep Learning là một thuật toán dựa trên một số ý tưởng từ não bộ tới việc tiếp thu nhiều tầng biểu đạt, cả cụ thể lẫn trừu tượng, qua đó làm rõ nghĩa của các loại dữ liệu.

Deep Learning được ứng dụng trong nhận diện hình ảnh, nhận diện giọng nói, xử lý ngôn ngữ tự nhiên.

Điểm đặc biệt của Deep Learning là tính chính xác dựa vào lượng dữ liệu, lượng dữ liệu có thể có kích thước khổng lồ mà không bị hạn chế.

**Code**

function Img = readAndPreprocessImage(filename)

I = imread(filename);

if ismatrix(I)

I = cat(3,I,I,I);

end

Img = imresize(I, [227 227]);

end

===================================================

function nImgTrue = deeplearning( )

net = alexnet()

featureLayer = 'fc7';

if exist('DataTrainAlexnet.mat','file') == 2

load('DataTrainAlexnet.mat');

else

strFolderDataTrain = fullfile('DataTrain');

categories = {'0', '1', '2','3','4','5','6','7','8','9'};

imdsDataTrain = imageDatastore(fullfile(strFolderDataTrain, categories), 'LabelSource', 'foldernames');

imdsDataTrain.ReadFcn = @(filename)readAndPreprocessImage(filename);

featuresDataTrain = activations(net, imdsDataTrain, featureLayer,'MiniBatchSize', 32, 'OutputAs', 'columns');

lblDataTrain = imdsDataTrain.Labels;

save('DataTrainAlexnet.mat','featuresDataTrain','lblDataTrain');

end

if exist('ClassifierAlexnet.mat','file') == 2

load('ClassifierAlexnet.mat');

else

classifier = fitcecoc(featuresDataTrain, lblDataTrain,'Learners', 'Linear', 'Coding', 'onevsall', 'ObservationsIn', 'columns');

save('ClassifierAlexnet.mat','classifier');

end

if exist('DataTestAlexnet.mat','file') == 2

load('DataTestAlexnet.mat');

else

strFolderDataTest = fullfile('DataTest');

categories = {'0', '1', '2','3','4','5','6','7','8','9'};

imdsDataTest = imageDatastore(fullfile(strFolderDataTest, categories), 'LabelSource', 'foldernames');

imdsDataTest.ReadFcn = @(filename)readAndPreprocessImage(filename);

featuresDataTest = activations(net, imdsDataTest, featureLayer, 'MiniBatchSize',32);

lblActualDataTest = imdsDataTest.Labels;

lblResult = predict(classifier, featuresDataTest);

save('DataTestAlexnet.mat','featuresDataTest','lblActualDataTest','lblResult');

end

nResult = (lblActualDataTest == lblResult);

nImgTrue = sum(nResult);

end

Kết quả nhận dạng đúng khi chạy với tập dữ liệu Test set images 10000 ảnh là 9901, chiếm 99.01%.

## **2.7. Giao diện chương trình**



**Mô tả các chức năng:**

**-** *Feature:* Chọn đặc trưng cần nhận dạng (RAW, HOG, LBP, BoW, Deep learning)

- *KNN/SVM:* Chọn bộ phân lớp (KNN hoặc SVM)

- *CellSize:* Chọn CellSize tương ứng ([2 2], [4 4], [8 8]), chỉ có trong đặc trưng HOG

- *NumNeighbors:* Chọn NumNeighbors tương ứng (từ 4 đến 24), chỉ có trong đặc trưng LBP

- *Radius:* Chọn Radius tương ứng (từ 1 đến 5), chỉ có trong đặc trưng LBP

- Nút *“Nhận dạng”* bấm để xem kết quả bao nhiêu ảnh đúng đối với bộ dữ liệu test 10000 ảnh.

Ô *“KẾT QUẢ NHẬN DẠNG ĐÚNG”* để xem kết quả các ảnh đúng sau khi bấm nút *“Nhận dạng”*

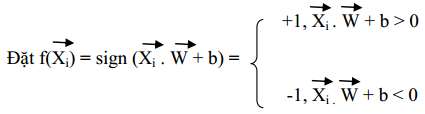
***Lưu ý:*** Do tập dữ liệu huấn luyện và test lớn nên chương trình cần thời gian thực thi để cho ra kết quả.

# **III. Tìm hiểu thuật toán Support Vector Machine (SVM)**

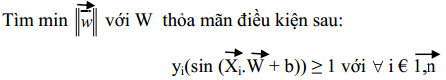
SVM thực chất là một bài toán tối ưu, mục tiêu của thuật toán này là tìm được một không gian F và siêu phẳng quyết định f trên F sao cho sai số phân loại là thấp nhất.

Cho tập mẫu (x1, y1), (x2, y2), … (xf, yf)} với xi ∈ Rn , thuộc vào hai lớp nhãn: yi ∈ {-1,1} là nhãn lớp tương ứng của các xi (-1 biểu thị lớp I, 1 biểu thị lớp II).

Ta có, phương trình siêu phẳng chứa vectơ trong không gian:



Như vậy, biểu diễn sự phân lớp của vào hai lớp như đã nêu. Ta nói yi =+1 nếu lớp I và yi = -1 nếu lớp II . Khi đó, để có siêu phẳng f ta sẽ phải giải bài toán sau:



Bài toán SVM có thể giải bằng kỹ thuật sử dụng toán tử Lagrange để biến  
đổi về thành dạng đẳng thức. Một đặc điểm thú vị của SVM là mặt phẳng quyết  
định chỉ phụ thuộc các Support Vector và nó có khoảng cách đến mặt phẳng quyết định là A close up of a sign

Description generated with very high confidence. Cho dù các điểm khác bị xóa đi thì thuật toán vẫn cho kết quả giống như ban đầu. Đây chính là điểm nổi bật của phương pháp SVM so với các phương pháp khác vì tất cả các dữ liệu trong tập huấn luyện đều được đùng để tối ưu hóa kết quả.

## **3.1. Phân hai lớp với SVM:**

Bài toán đặt ra là: Xác định hàm phân lớp để phân lớp các mẫu trong tương lai, nghĩa là với một mẫu dữ liệu mới xi thì cần phải xác định xi được phân vào lớp +1 hay lớp -1.

Để xác định hàm phân lớp dựa trên phương pháp SVM, ta sẽ tiến hành tìm hai siêu phẳng song song sao cho khoảng cách *y* giữa chúng là lớn nhất có thể để phân tách hai lớp này ra làm hai phía. Hàm phân tách tương ứng với phương trình siêu phẳng nằm giữa hai siêu phẳng tìm được

A close up of a device

Description generated with high confidence

*Hình 3.1.1: Phân 2 lớp bằng SVM*

***Các điểm mà nằm trên hai siêu phẳng phân tách được gọi là các Support Vector***. Các điểm này sẽ quyết định đến hàm phân tách dữ liệu.

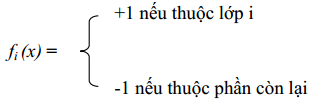
## **3.2. Phân nhiều lớp với SVM**

Để phân nhiều lớp thì kỹ thuật SVM nguyên thủy sẽ chia không gian dữ liệu thành 2 phần và quá trình này lặp lại nhiều lần. Khi đó hàm quyết định phân dữ liệu vào lớp thứ i của tập n , 2-Iớp sẽ là:

A picture containing thing, object, clock

Description generated with very high confidence

Những phần tử *x* là support vector sẽ thỏa điều kiện



Như vậy, bài toán phân nhiều lớp sử dụng phương pháp SVM hoàn toàn có thể thực hiện giống như bài toán hai lớp. Bằng cách sử dụng chiến lược *"một- đốimột”*(*one - against - one*). Giả sử bài toán cần phân loại có k lớp (k > 2), chiến lược *"một-đối-một”*sẽ tiến hành k(k-l)/2 lần phân lớp nhị phân sử dụng phương pháp SVM. Mỗi lớp sẽ tiến hành phân tách với k-1 lớp còn lại để xác định k-1 hàm phân tách dựa vào bài toán phân hai lớp bằng phương pháp SVM.

## **3.3. Các bước chính của SVM**

Phương pháp SVM yêu cầu dữ liệu được diễn tả như các vector của các số thực. Như vậy nếu đầu vào chưa phải là số thì ta cần phải tìm cách chuyển chúng về dạng số của SVM.

Tiền xử lý dữ liệu: Thực hiện biến đổi dữ liệu phù hợp cho quá trình tính toán, tránh các số quá lớn mô tả các thuộc tính. Thường nên co giãn (*scaling*) dữ liệu để chuyển về đoạn [-1, 1] hoặc [0, 1].

Chọn hàm hạt nhân: Lựa chọn hàm hạt nhân phù hợp tương ứng cho từng bài toán cụ thể để đạt được độ chính xác cao trong quá trình phân lớp.

Thực hiện việc kiểm tra chéo để xác định các thám số cho ứng đụng. Điều này cũng quyết định đến tính chính xác của quá trình phân lớp.

Sử dụng các tham số cho việc huấn luyện với tập mẫu. Trong quá trình huấn luyện sẽ sử dụng thuật toán tối ưu hóa khoảng cách giữa các siêu phẳng trong quá trình phân lớp, xác định hàm phân lớp trong không gian đặc trưng nhờ việc ánh xạ dữ liệu vào không gian đặc trưng bằng cách mô tả hạt nhân, giải quyết cho cả hai trường hợp dữ liệu là phân tách và không phân tách tuyến tính trong không gian đặc trưng.

Kiểm thử tập dữ liệu Test.

# **TÀI LIỆU THAM KHẢO**

[1] TS. Lê Đình Duy, *Bài giảng Nhận dạng thị giác và ứng dụng*, 2017.

[2] TS. Nguyễn Tấn Trần Minh Khang, *Bài giảng lập trình Matlab*, 2017.

[3] Phạm Văn Sơn, *Tìm hiểu về Support vector machine cho bài toán phân lớp quan điểm*, 2012.

[4] Nguyễn Trọng Nhân, *Tối ưu hóa không gian lưu trữ đặc trưng ảnh.*

[5] TS. Nguyễn Hữu Tuân, ThS. Nhuyễn Văn Thủy, *Xây dựng hệ thống nhận dạng mặt tự động sử dụng LPQ*, 2016