**MỞ ĐẦU**

Thị giác máy tính (computer vision) là một lĩnh vực liên ngành thuộc ngành khoa học máy tính với mục tiêu làm cho máy tính có thể đạt được sự hiểu biết cấp cao từ hình ảnh kĩ thuật số hoặc video. Từ quan điểm kĩ thuật, là làm sao để máy tính có thể thực hiện được một cách tự động các chức năng mà hệ thống thị giác của con người có thể làm.

Nhận dạng thị giác là một trong những lĩnh vực của thị giác máy tính. Với mong muốn máy tính có thể thông minh như con người thì việc huấn luyện máy tính có thể nhận dạng được đúng các hình ảnh hoặc video là một bài toán lớn, được nhiều nhà nghiên cứu quan tâm.

Ngày nay với sự phát triển mạnh mẽ của kĩ thuật, các GPU tầm trung đã có thể có thể xử lí đến hàng teraflops ( GPU tầm trung NVidia GTX 1070 có thể xử lí 6.5 TFLOPS – 6.5 nghìn tỉ phép tính trên giây). Từ đó các nhà nghiên cứu đã có thể tiếp cận và đưa ra các thuật toán Deep Learning giúp cho độ chính xác của việc nhận diện thị giác lên cao, và tốc độ xử lí đạt được ở mức thời gian thực.

**I. Giới thiệu**

Nhận dạng thị giác là một trong những lĩnh vực của thị giác máy tính. Với mong muốn máy tính có thể thông minh như con người thì việc huấn luyện máy tính có thể nhận dạng được đúng các hình ảnh hoặc video là một bài toán lớn, được nhiều nhà nghiên cứu quan tâm.

**II. Tình hình nghiên cứu**

Nhận dạng thị giác là một bài toán cổ điển (classical problem) của ngành Computer Vision. Có thể chia nhận dạng thị giác thành các bài toán nhỏ như bên dưới:

* **Phân loại đối tượng** (*Object classification*), là phân loại một hay nhiều đối tượng đã được học trước.
* **Nhận dạng** (*Identification*), là việc nhận dạng một người hay một đối tượng xác định (VD: nhận dạng khuôn mặt hay dấu vân tay của một người cụ thể)
* **Phát hiện** (*Detection*), dữ liệu đầu vào được xử lí cho một số tình huống cụ thể (phát hiện những tế bào bất thường trong ảnh chụp y tế)

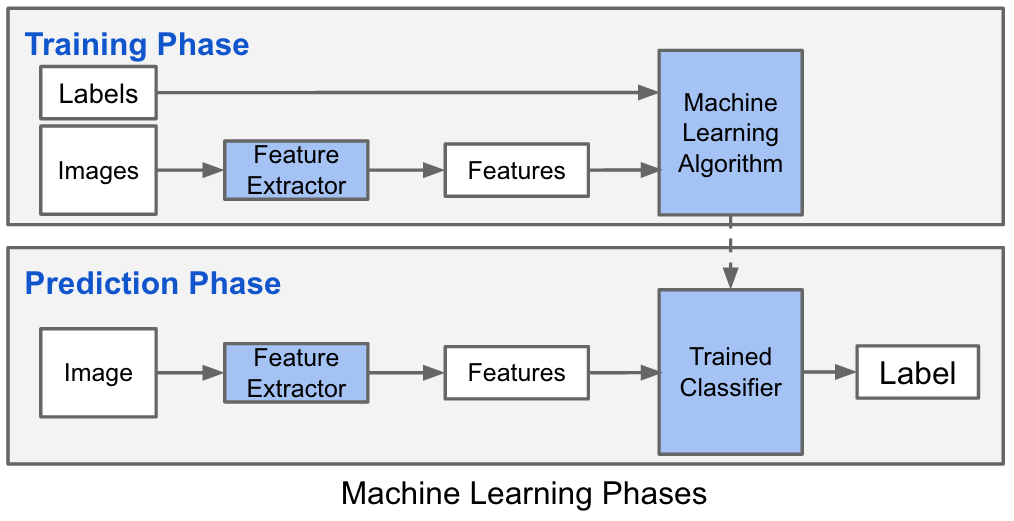
Hiện tại, các thuật toán tốt nhất cho những bài toán trên thì dựa vào convolutional neural networks (CNN). CNN gây được sự chú ý và trở thành thuật toán đầu tiên được nghĩ đến trong nhận dạng thị giác là vào năm 2010, AlexNet, một model CNN đã chiến thắng một cách áp đảo về độ chính xác so với các model truyền thống khác.

Tuy nhiên việc nhận dạng bằng CNN cũng có điểm yếu là thường không nhận dạng được các ảnh đã xử lí về màu sắc, độ méo, độ nghiêng, … - những điều không làm khó con người. Tuy nhiên CNN lại mạnh mẽ hơn con người là có thể dễ dàng học để phân lớp được những đối tượng vào lớp có độ mịn (fine-grained classes) như chủng loại chó (chó phốc, chó bull, chó bẹc-giê, …) hay loại cá (cá ngừ, cá bớp, cá trắm, cá hồi,…)

**III. Phương pháp**

Các phương pháp sẽ tập trung vào việc nhận dạng các pattern và quy luật của data train. Các data train ở đây sẽ được đánh nhãn. Như vậy các phương pháp nhận dạng sẽ là một tập hợp các phương pháp học có giám sát (supervised learning). Do đó hầu hết các trường hợp việc nhận dạng có thể xem như tương đồng với học máy (machine learning). Ngoài ra từ năm 2010, sau khi một model CNN là Alexnet chiến thắng thuyết phục ở ImageNet Challenge thì tiềm năng của mạng neural network đã được công nhận. Các kĩ thuật nhận dạng hiện đại nhất chủ yếu sử dụng Deep Learning

**1/ Machine Learning**



*Hình 1: mô hình tổng quát Machine Learning*

Ở mô hình các bước của Machine Learning như trên thì ngoài việc chuẩn bị data Train và Test thì 2 bước quan trọng nhất là trích xuất đặc trưng (feature extractor) và machine learning algorithm.

**a. Một số phương pháp trích xuất đặc trưng:**

* *Phương pháp HOG - Histogram of Oriented Gradients*:

HOG là một phương pháp trích xuất đặt trưng được sử dụng trong Computer Vision để detect object. Phương pháp HOG dựa trên việc đếm số lần xuất hiện của 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 một bức ảnh thành các vùng con được gọi là các “tế bào” (cells) 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 và cuối cùng là chuẩn hóa các khối.

**Tính gradient**: Đây là bước đầu tiên trong phương pháp HOG và thường được thực hiện bằng hai phép nhân chập ảnh gốc với 2 nhân 1 chiều tương ứng với các toán tử lấy đạo hàm ảnh theo hai hướng Ox và Oy, cụ thể là:

Dx = [-1 0 1] và Dy = [1 0 -1]T,

trong đó T là ký hiệu của phép chuyển vị ma trận.

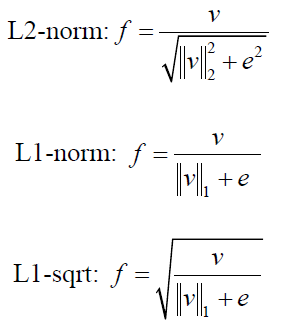
Với một ảnh input I, ta sẽ có 2 ảnh đạo hàm riêng theo 2 hướng được tính bằng các công thức: Ix = I\*Dx và Iy = I\*Dy.

Thành phần cường độ (magnitude) và hướng (orientation) sẽ được tính theo các công thức: |G| = sqrt(Ix + Iy ) và =arctan(Iy, Ix). Ví dụ về kết quả của phép lấy đạo hàm có thể xem ở hình sau:

*Hình 2: Ảnh input và hai đạo hàm của nó.*

**Gán hướng:** Sau khi tính đạo hàm, ta có thành phần hướng ở các điểm ảnh trong mỗi cell của ảnh input. Các giá trị hướng này được chia thành các vùng hướng đều nhau trong khoảng từ 0 tới 360o. Theo các tác giả của HOG, thì 9 vùng chia cho kết quả tốt nhất đối với các bài toán nhận dạng. Có nghĩa là các giá trị hướng nằm trong khoảng [0, 360/9) sẽ được gán bằng 1 (coi là như nhau), các giá trị hướng trong khoảng [360/9, 2\*360/9) sẽ được gán bằng 2, vân vân. Sau đó histogram của mỗi cell sẽ được tính độc lập.

**Tính histogram của các khối**: Để đáp ứng tốt hơn với các thay đổi về điều kiện ánh sáng và độ tương phản, các giá trị gradient cần phải được chuẩn hóa cục bộ, điều này đòi hỏi việc nhóm các cell thành các vùng liên kết không gian với nhau gọi là các khối. Vector HOG do đó sẽ là tổng hợp các histogram của các cell đã được chuẩn hóa từ tất cả các khối. Các khối sẽ được chia và không tách biệt nhau, nói cách khác mỗi cell sẽ đóng góp thông tin về các hướng (tại các điểm ảnh của nó) nhiều hơn 1 lần.

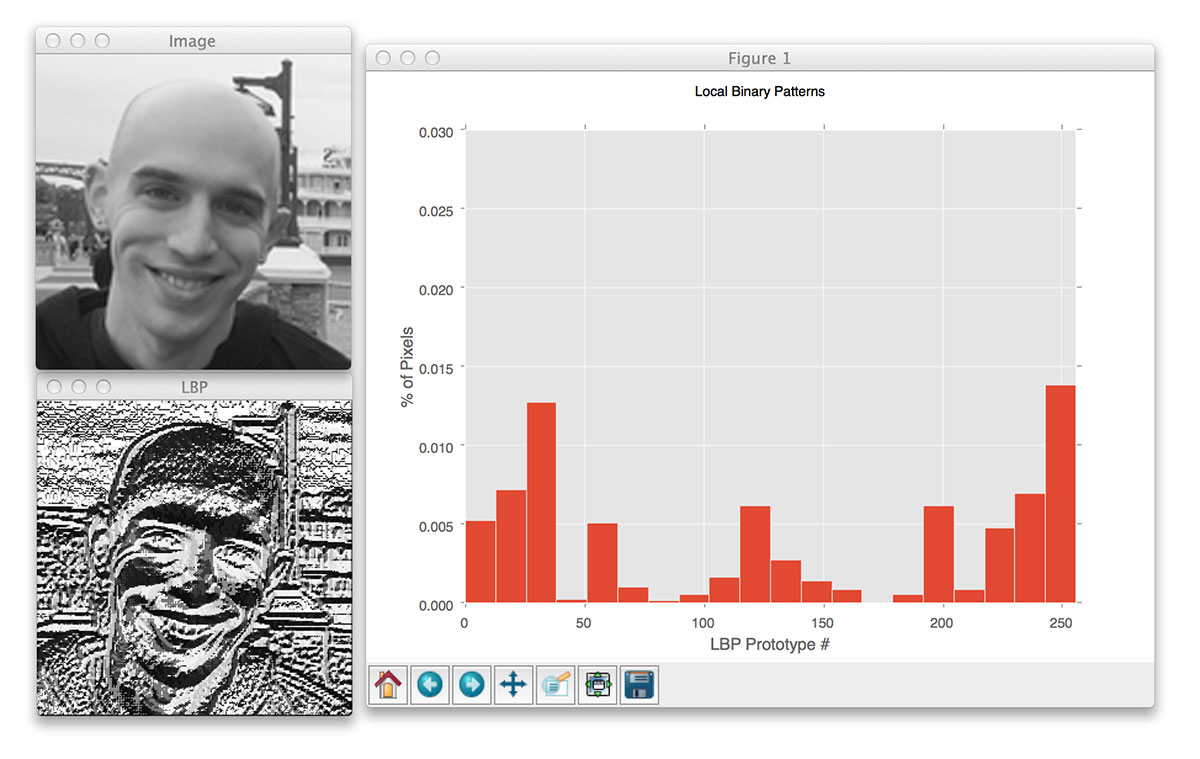
**Chuẩn hóa khối**: Có nhiều phương pháp có thể được dùng cho việc chuẩn hóa khối. Gọi v là vector cần chuẩn hóa chứa tất cả các histogram của một khối, ||vk|| là giá trị chuẩn của nó theo các chuẩn k=1, 2 và e là một hằng số nhỏ, khi đó các giá trị chuẩn hóa có thể tính bằng 1 trong các công thức sau:

* *Phương pháp LBP - Histogram of Oriented Gradients*:

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. Hình dưới minh họa cách tính độ tương phản trực giao (C) là hiệu cấp độ xám trung bình của các điểm ảnh lớn hơn hoặc bằng ngưỡng với các điểm ảnh thấp hơn ngưỡng.



*Hình 3. Ví dụ về LBP và độ tương phản cục bộ C.*

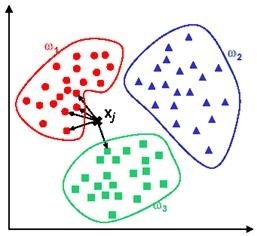


*Hình 4: Hình ảnh input và output khi sử dụng LBP*

**b. Một số phương thuật toán máy học:**

* *Phương pháp người láng giềng k gần nhất (k-nearest neighbors algorithm (k-NN))*:

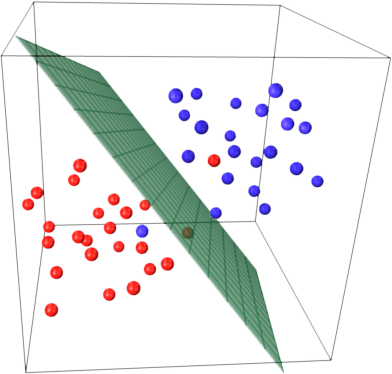
Ý tưởng của thuật toán là giả sử dữ liệu được biểu diễn với dạng các phần tử trong không gian, thì có thể xem như các phần tử được phân bổ gần nhau sẽ là láng giềng của nhau và có khoảng cách không gian gần nhau. Từ đó ta có thể so sánh một phần tử với k phần tử gần nhất. Nếu các k phần tử gần nhất có thuộc tính, tính chất gì, ta có thể dự đoán phần từ được so sánh có tính chất đó.



*Hình 5: Ví dụ về việc kiểm tra xj trong không gian 2 chiều, sử dụng k-nn với k = 5. Ta nhận thấy xj gần với 4 phần từ ở group đỏ so với 1 phần từ ở group xanh. Từ đó kết luận được xj có tính chất giống với các phần tử thuộc group đỏ*

* *Phương pháp máy vec-tơ hỗ trợ (support vector machines algorithm (SVMs, hay support vector networks))*:

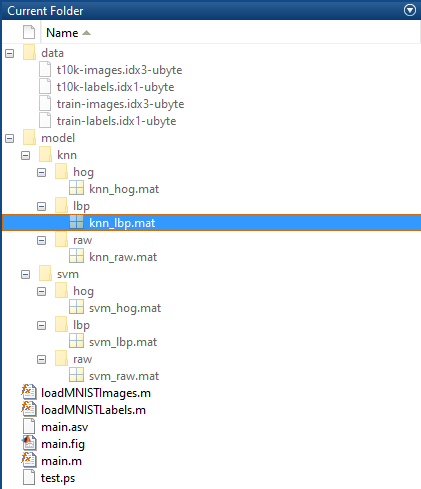
Giả sử dữ liệu được biểu diễn với dạng các phần tử trong không gian, ta có thể tìm được một siêu phẳng để chia cắt thành hai phần không gian để có phân chia các phần tử thành hai nhóm phần tử có tính chất nhất định khác nhau. Từ đó với một phần bất kì, kiểm tra xem nó thuộc phần không gian nào đã được siêu phẳng chia cắt. Từ đó có thể xác định được tính chất của phần tử này.



*Hình 6: Siêu phẳng xanh lá đã cắt không gian thành phần với phần bên trái phần lớn là phần từ màu đỏ, bên phải phần lớn là các phần từ màu xanh. Từ đó, với phần phần tử bất kì thêm vào, ta kiểm tra xem nó thuộc phần bên nào của siêu phẳng để dự đoán được tính chất của nó là màu đỏ hay xanh.*

**c. Ứng dụng Machine Learning cho bài toán MNIST**

Từ các phương pháp trích xuất đặc trưng thì em sẽ xây dựng một ứng dụng mẫu sử dụng các phương pháp machine learning đã giới thiệu ở trên để xây dựng một ứng dụng nhận dạng chữ viết tay.



*Hình 7 cấu trúc thư mục của project*

Dữ liệu của bài toán nhận dạng chữ viết tay được lấy từ:

<http://yann.lecun.com/exdb/mnist/index.html>

Sau khi get dữ liệu thì ta sẽ có 4 file gồm hình ảnh và nhãn của data train và data test ở thư mục data.

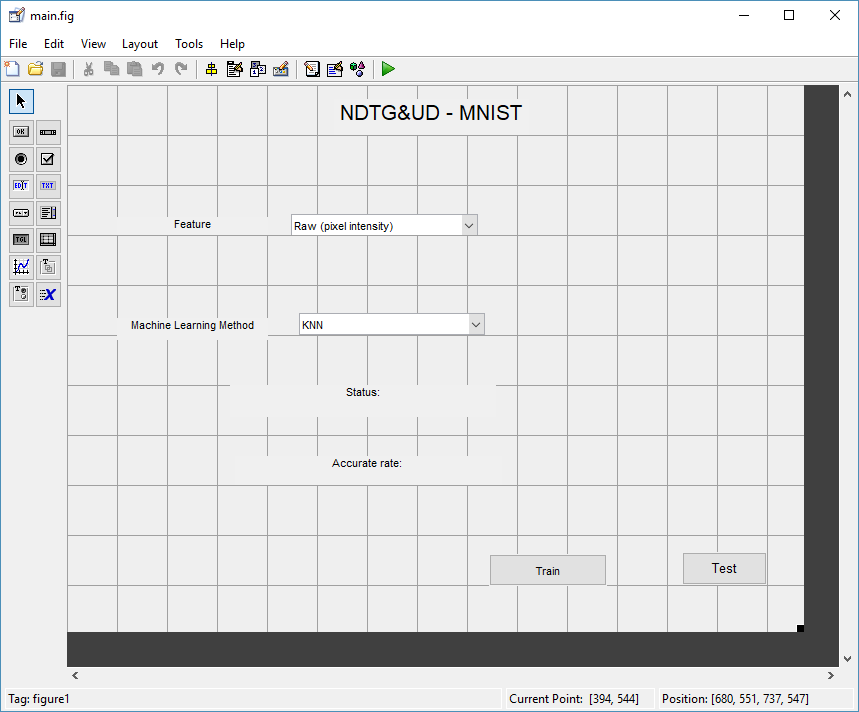
Ở thư mục model là nơi lưu trữ các model tương ứng khi thự hiện train xong. Việc lưu trữ này nhằm mục đích load model lên khi thực hiện test.

Ngoài ra về việc load ảnh và nhãn của bộ dữ liệu MNIST có sử dụng câu lệnh được phát triển bởi đại học Standford theo đường dẫn bên dưới:

<http://ufldl.stanford.edu/wiki/resources/mnistHelper.zip>

Chương trình chính gồm 2 file bào gồm file main.fig chứ các thành phần GUI và file main.h chứa các thành phần thực hiện chương trình

File main.fig



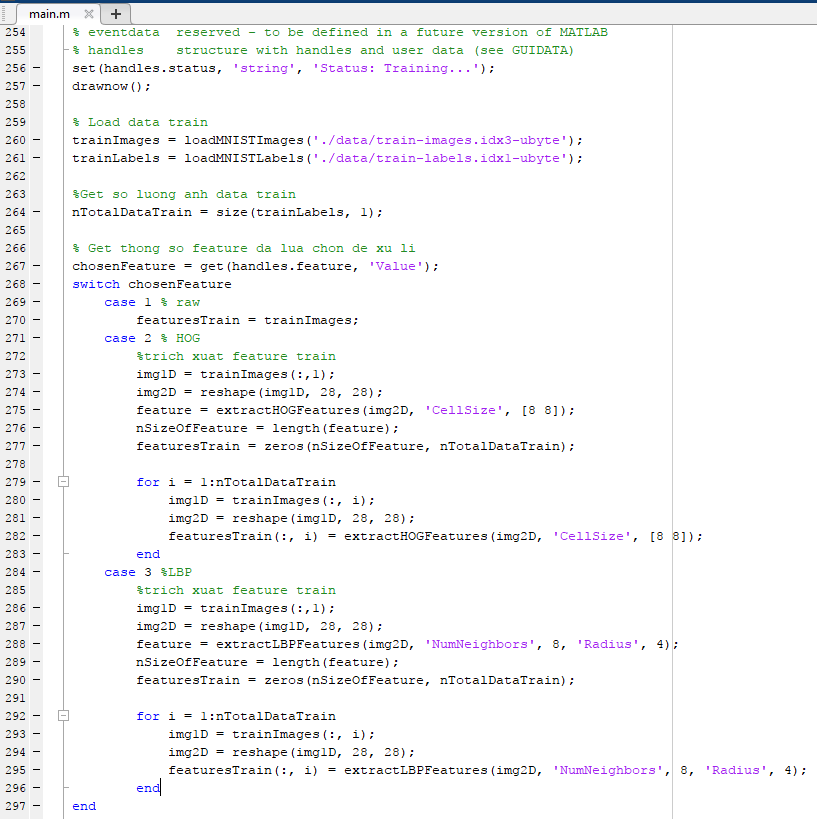
File main.h:

Khi tạo xong file GUI thì matlab sẽ tự động tạo 1 file .m có tên giống với tên file GUI với các fuction rỗng được xây dựng sẵn để người dùng implement.

Ở GUI này sẽ 2 popup cho phép người dùng chọn đặc trưng (raw, HOG, LBP) và chọn phương pháp ML (KNN, SVM)

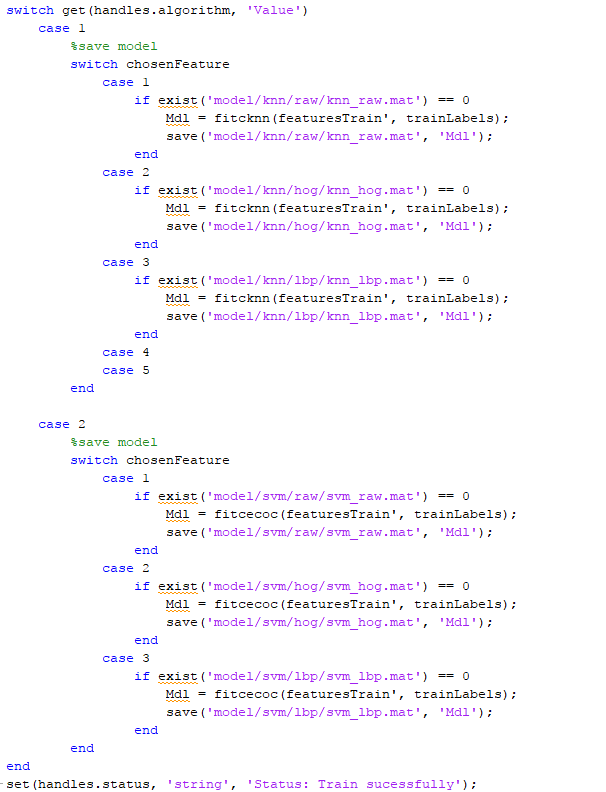
Tiếp đến sẽ có 2 nút Train và Test tương ứng cho việc thực hiện train và test tương ứng với đặc trưng và thuật toán ML người dùng đã chọn

*Thực hiện train:*

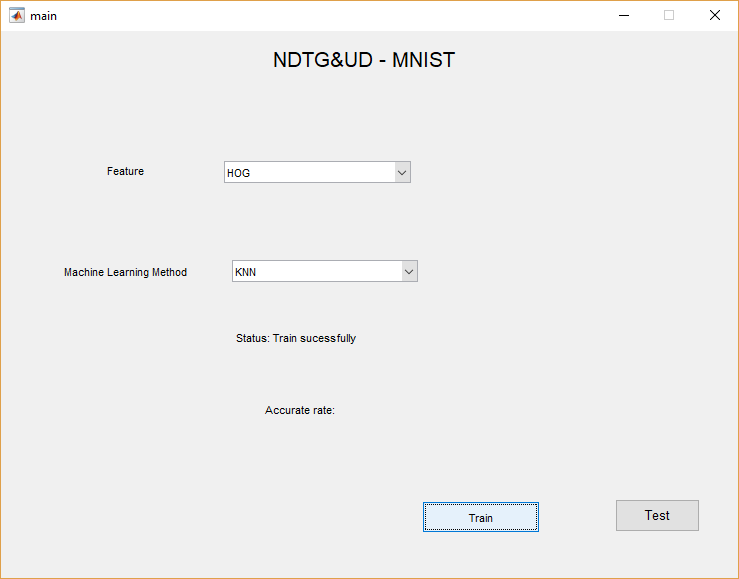


Khi thực hiện train, sẽ thay đổi status để người dùng biết ứng dụng đang thực hiện việc gì.

Sau đó sẽ load bộ dữ liệu train lên, từ feature đã được lựa chọn ở GUI, ta sẽ thực hiện trích xuất model tương ứng. Ở demo thì em có thêm một số option khi trích xuất đặc trưng. HOG em sử dụng thêm thông số Cellsize là 8x8. LBP em sử dụng NumNeighbors là 8 và Radius là 4.



Sau khi đã trích xuất xong đặc trưng, dựa vào các option người dùng đã chọn em thực hiện xây dựng Model, sau khi xây dựng xong sẽ save lại vào các thư mục con tương ứng trong thư mục model. Cuối cùng thông báo cho người dùng quá trình train đã thành công.



*Thực hiện test:*

Souce code:

% --- Outputs from this function are returned to the command line.

function varargout = main\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on button press in testbutton.

function testbutton\_Callback(hObject, eventdata, handles)

% hObject handle to testbutton (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

set(handles.status, 'string', 'Status: Testing...');

drawnow();

% Load data test

testImages = loadMNISTImages('./data/t10k-images.idx3-ubyte');

testLabels = loadMNISTLabels('./data/t10k-labels.idx1-ubyte');

% get so luong data test

nTotalDataTest = size(testLabels, 1);

%Load saved model

switch get(handles.algorithm, 'Value')

case 1

%save model

switch get(handles.feature, 'Value')

case 1

if exist('model/knn/raw/knn\_raw.mat')

% load pretrained model

load('model/knn/raw/knn\_raw.mat');

featuresTest = testImages;

else

error('Please train before test');

end

case 2

if exist('model/knn/hog/knn\_hog.mat')

% load pretrained model

load('model/knn/hog/knn\_hog.mat');

img1D = testImages(:,1);

img2D = reshape(img1D, 28, 28);

feature = extractHOGFeatures(img2D, 'CellSize', [8 8]);

nSizeOfFeature = length(feature);

featuresTest = zeros(nSizeOfFeature, nTotalDataTest);

for i = 1:nTotalDataTest

img1D = testImages(:, i);

img2D = reshape(img1D, 28, 28);

featuresTest(:, i) = extractHOGFeatures(img2D, 'CellSize', [8 8]);

end

else

error('Please train before test');

end

case 3

if exist('model/knn/lbp/knn\_lbp.mat')

% load pretrained model

load('model/knn/lbp/knn\_lbp.mat');

%trich xuat feature test

img1D = testImages(:,1);

img2D = reshape(img1D, 28, 28);

feature = extractLBPFeatures(img2D, 'NumNeighbors', 8, 'Radius', 4);

nSizeOfFeature = length(feature);

featuresTest = zeros(nSizeOfFeature, nTotalDataTest);

for i = 1:nTotalDataTest

img1D = testImages(:, i);

img2D = reshape(img1D, 28, 28);

featuresTest(:, i) = extractLBPFeatures(img2D, 'NumNeighbors', 8, 'Radius', 4);

end

else

error('Please train before test');

end

end

%danh gia

resultLabels = predict(Mdl, featuresTest');

case 2

%save model

switch get(handles.feature, 'Value')

case 1

if exist('model/svm/raw/svm\_raw.mat')

% load pretrained model

load('model/svm/raw/svm\_raw.mat');

featuresTest = testImages;

else

error('Please train before test');

end

case 2

if exist('model/svm/hog/svm\_hog.mat')

% load pretrained model

load('model/svm/hog/svm\_hog.mat');

img1D = testImages(:,1);

img2D = reshape(img1D, 28, 28);

feature = extractHOGFeatures(img2D, 'CellSize', [8 8]);

nSizeOfFeature = length(feature);

featuresTest = zeros(nSizeOfFeature, nTotalDataTest);

for i = 1:nTotalDataTest

img1D = testImages(:, i);

img2D = reshape(img1D, 28, 28);

featuresTest(:, i) = extractHOGFeatures(img2D, 'CellSize', [8 8]);

end

else

error('Please train before test');

end

case 3

if exist('model/svm/lbp/svm\_lbp.mat')

% load pretrained model

load('model/svm/lbp/svm\_lbp.mat');

%trich xuat feature test

img1D = testImages(:,1);

img2D = reshape(img1D, 28, 28);

feature = extractLBPFeatures(img2D, 'NumNeighbors', 8, 'Radius', 4);

nSizeOfFeature = length(feature);

featuresTest = zeros(nSizeOfFeature, nTotalDataTest);

for i = 1:nTotalDataTest

img1D = testImages(:, i);

img2D = reshape(img1D, 28, 28);

featuresTest(:, i) = extractLBPFeatures(img2D, 'NumNeighbors', 8, 'Radius', 4);

end

else

error('Please train before test');

end

end

%danh gia

resultLabels = predict(Mdl, featuresTest');

end

rightPredict = sum(resultLabels == testLabels);

rightPredictRate = (rightPredict/nTotalDataTest)\*100;

set(handles.status, 'string', 'Status: Test successfully');

set(handles.resulttext, 'string', ['Accurate rate: ', num2str(rightPredictRate), '%']);

% --- Executes on selection change in feature.

function feature\_Callback(hObject, eventdata, handles)

Source code trên được gọi khi bắt đầu nhấn nút test.

Đầu tiên sẽ thực hiện thay đối trạng thái để người dùng biết ứng dụng đang thực thi, sau đó load datatest.

Dựa và các option người dùng chọn ở GUI, chương trình sẽ load các model được lưu ở thư mực tương ứng, tiếp theo là trích xuất đặc trưng của datatest và thực hiện predict.

Cuối cùng khi predict xong sẽ hiển thị tỉ lệ chính xác của thuật toán và thay đổi trạng thái test thành công

**2/ Deep Learning**

1. **Giới thiệu**

Deep learning hiện đang là công nghệ machine learning hot nhất hiện giờ, kể cả trong giảng đường nghiên cứu hoặc trong các công ty trên thị trường. Các công ty mạnh nhất của công nghiệp phần mềm và mạng Internet như Google, Microsoft, Facebook đều nắm trong tay những công nghệ deep learning quy mô lớn nhất. Giới nghiên cứu cũng cực kì quan tâm về deep learning. Đơn cử là việc người ta thấy các conference (hội thảo) cũ về machine learning không còn đủ chỗ để chứa các paper (bài báo) về deep learning nữa nên đã lập ra riêng một số conference như ICLR chỉ chuyên về deep learning.

Deep learning thường được gắn với cụm từ neural network. Neural network hoàn toàn không mới, thậm chí chúng là những model machine learning đầu tiên xuất hiện từ những năm 1950 khi machine learning mới phôi thai. Tuy nhiên, khi ấy, dữ liệu chưa nhiều, máy tính chưa mạnh nên những model này không thể sánh kịp với những model đơn giản như logistic regression hay support vector machines. Yann LeCunn, hiện giờ là Director of Research tại Facebook, từng khốn đốn trong thời gian dài vì các paper về deep learning của nhóm ông liên tục bị từ chối bởi các conference lớn. Đến nỗi, ông viết một bức thư gửi cho ban tổ chức một conference thuộc dạng lớn nhất trong giới Computer Vision, bảo là tôi sẽ không bao giờ gửi paper đến cho conference này, trừ phi các ngài xóa bỏ định kiến về deep learning.

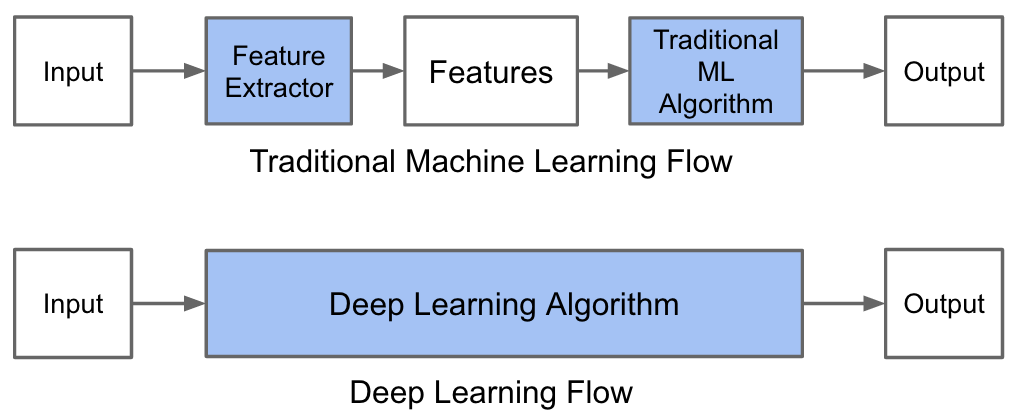
Deep learning muốn phát triển được cần ba yếu tố sau:

1. Lượng dữ liệu dồi dào.

2. Tốc độ tính toán của máy tính nhanh hơn.

3. Các kỹ thuật nâng cao để huấn luyện neural network ra đời.

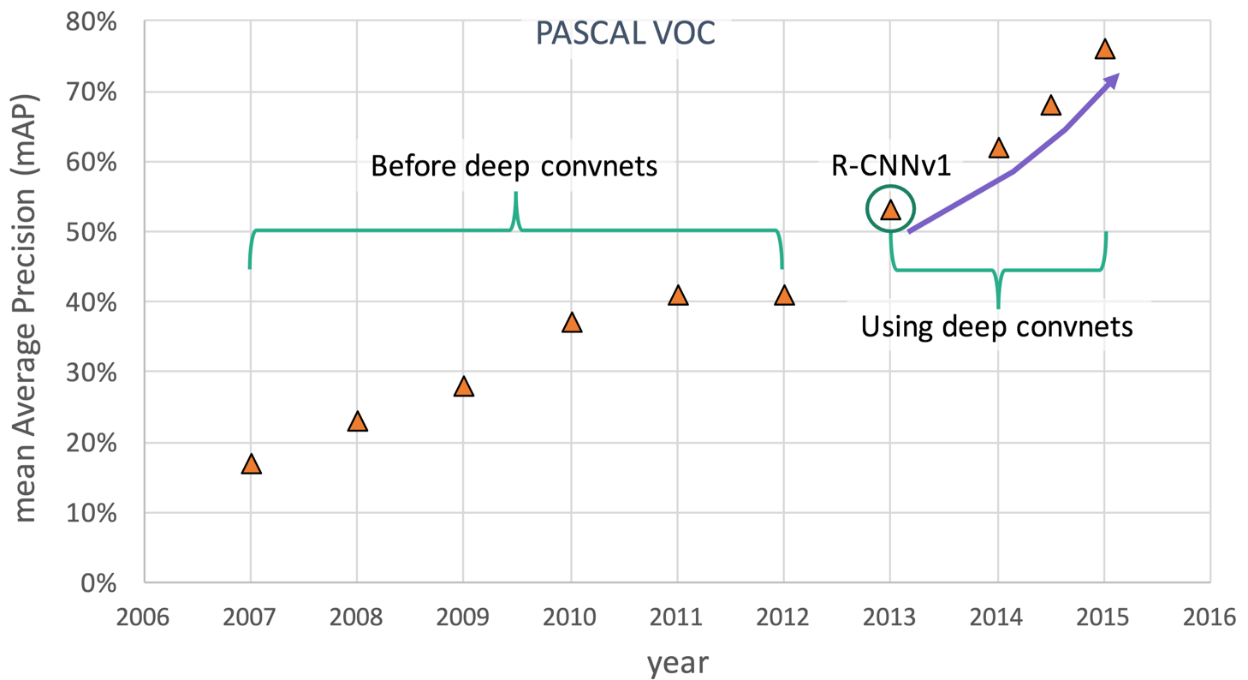
Dòng chảy của lịch sử khiến 2 yếu tố đầu dần dần đạt được. Mãi đến năm 2006, bài báo của Geoff Hinton trên tạp chí Nature gây được sự chú ý lớn trong cộng đồng khoa học. Năm 2010, AlexNet, một model convolutional neural network, chiến thắng ImageNet Challenge (một cuộc thi về nhận diện đồ vật trong hình ảnh) một cách ngoạn mục, bỏ xa các model truyền thống về độ chính xác. Từ đó người ta nhận ra được tiềm năng của neural network và dần dần hoàn thiện kỹ thuật huấn luyện chúng. Khi 3 yếu tố kể trên được hội tụ, kỷ nguyên deep learning được mở ra.



*Hình 7: Mô hình luồng xử lí của ML truyền thống và DL*

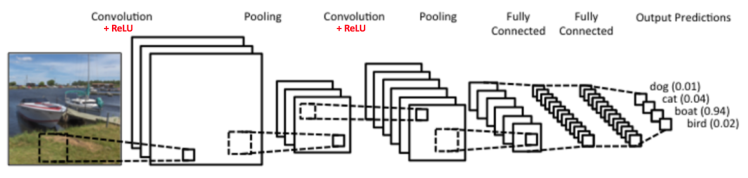
Đặc biệt với các bài toán nhận dạng ảnh, phát hiện đối tượng, etc - sử dụng deep learning có thể tăng độ chính xác lên hơn 10%.

Có nhiều thư viện về deep learning, do đó nếu hiểu được cách dùng, có thể dễ dàng tạo được các ứng dụng trong lĩnh vực nhận dạng với độ chính xác cao.



*Hình 8: biểu độ độ chính xác trước và sau khi có DL khi chạy trên bộ dữ liệu VOC*

Mô hình được sử dụng chủ yếu để nhận dạng thì giác bằng Deep Learning chủ yếu là sử dụng mạng Convolutional Neural Networks (CNNs)



*Hình 9: một mô hình CNN*

Khái niệm:

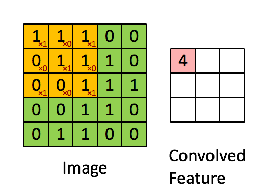
Convolution Feature/activation map/feature map là một map được tạo bằng cách trượt ma trận filter với ảnh để tính tích vô hướng.

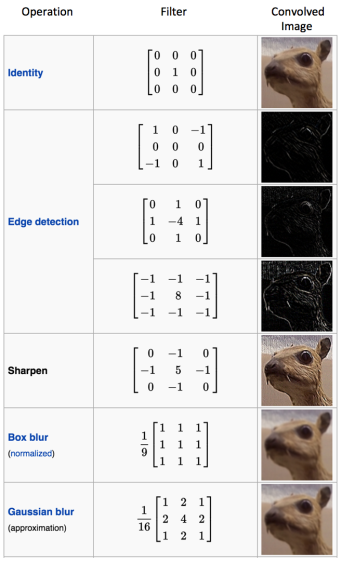
VD:

Filter



Tính toán





*Hình 10: Một số filter thông dụng*

Sau khi có được feature map thì cần phải thực hiện phi tuyến tính map này, vì dữ liệu thực tế thường là phi tuyến tính. Các phướng pháp phi tuyến tính thông dung là ReLU, Sigmoid, tanh... Tuy nhiên phương pháp được khuyến cáo sử dụng chính là ReLU

Công thức ReLU

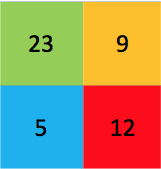
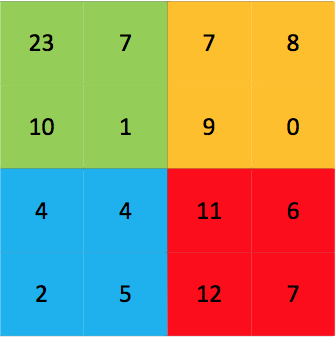


Pooling

Sử dụng để làm giảm độ phân giải của feature map mà vẫn giữ lại được các thông tín quan trọng.

Các Pooling có các dạng như: Max, Average, Sum etc.

Qua thực tế thì Max pooling thường cho kết quả tốt nhất.



2x2 pooling, stride 2

Max pooling

Average pooling

*Hình 11: Input và output khi sử dụng max pooling và average pooling với stride là 2*

Fully Connected Layer (FC)

Thường được sử cuối cùng của mạng CNN, mục đích dùng để phân lớp các feature đã get được ở các bước trên, từ đó có thể phấn lớp cho input images.

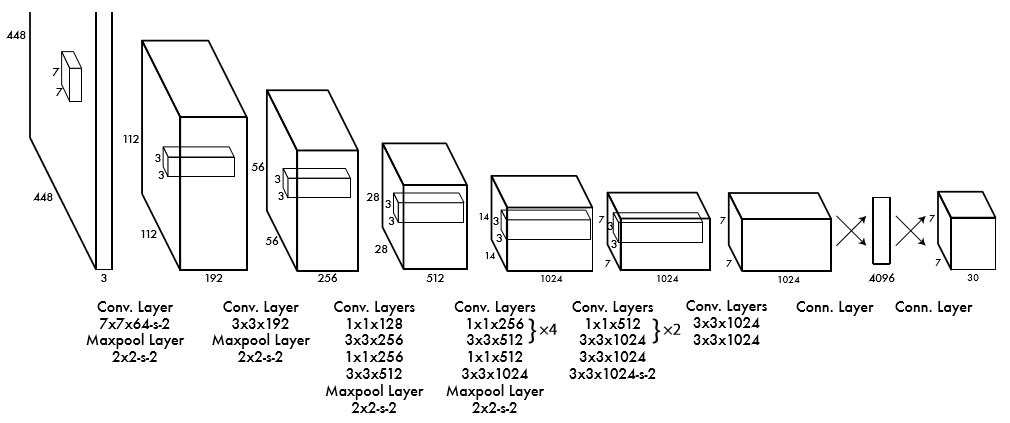
Sử dụng FC là không bắt buộc nhưng với chi phí thấp và thường cho kết quả tốt hơn là không sử dụng.

1. **Tìm hiểu và thử nghiệm phương pháp YOLO**

Phương pháp YOLO là một phương pháp Object Detection sử dụng mạng DL có tốc độ xử lí rất nhanh, có thể áp dụng vào thời gian thực như detect object trong video. Bài báo về phương pháp được publish vào năm 2016 tại CVPR.

Bên dưới là link của dự án YOLO

<https://pjreddie.com/darknet/yolo/>



*Hình 12: Mô hình mạng YOLO*

Nhóm tác giả đề xuất phương pháp tiếp cận bằng cách sử dụng một mạng neural network cho ảnh input, mạng này sẽ chia hình ảnh thành nhiều vùng sau đó dự đoán bounding boxes và xác suất cho từng vùng. Các các bounding boxes này được tính trọng số dựa vào xác suất đã được dự đoán.

Mô hình này có nhiều ưu điểm so với các hệ thống dựa trên classifier. Nó sẽ xử lí toàn bộ hình ảnh ở thời điểm test, do dó việc việc dự đoán sẽ thực hiện cho toàn bộ ảnh. Ngoài ra do chỉ sử dụng một mạng neural network khi dự đoán khiến cho tốc độ xử lí của phương pháp này nhanh hơn 1000x so với R-CNN và 100x so với Fast R-CNN

* Mô tả quá trình học viên đã chạy thử phương pháp này.

Ở đây em sử dụng server ubuntu phiên bản 16.04 với 2 core và 7.5 GB RAM

Clone source đường dẫn bên dưới

<https://github.com/pjreddie/darknet>

Sau khi clone xong source vào thư mục vừa clone chạy câu lệnh make để build source. ở đây sử dụng source C và python nên khi build sẽ cần install thêm một số thư việc, kiểm tra từng dòng lỗi báo thiếu thư viện nào, downloaf tuần tự đến khí có đủ thư viện và build thành công.

Để training cho mạng yolo thì tác giả đã sử dụng bộ data Pascal VOC từ 2007 đến 2012.

Do máy tính ubuntu của em không có GPU nên việc train bị crash giữa chừng vì mất session với server. Vì vậy em download pre-trained model tác giả có cung cấp theo đường dẫn bên dưới

<https://pjreddie.com/media/files/yolo.weights>

Sau khi đã có được model pretrained, ta đã có thể bắt đầu predict.

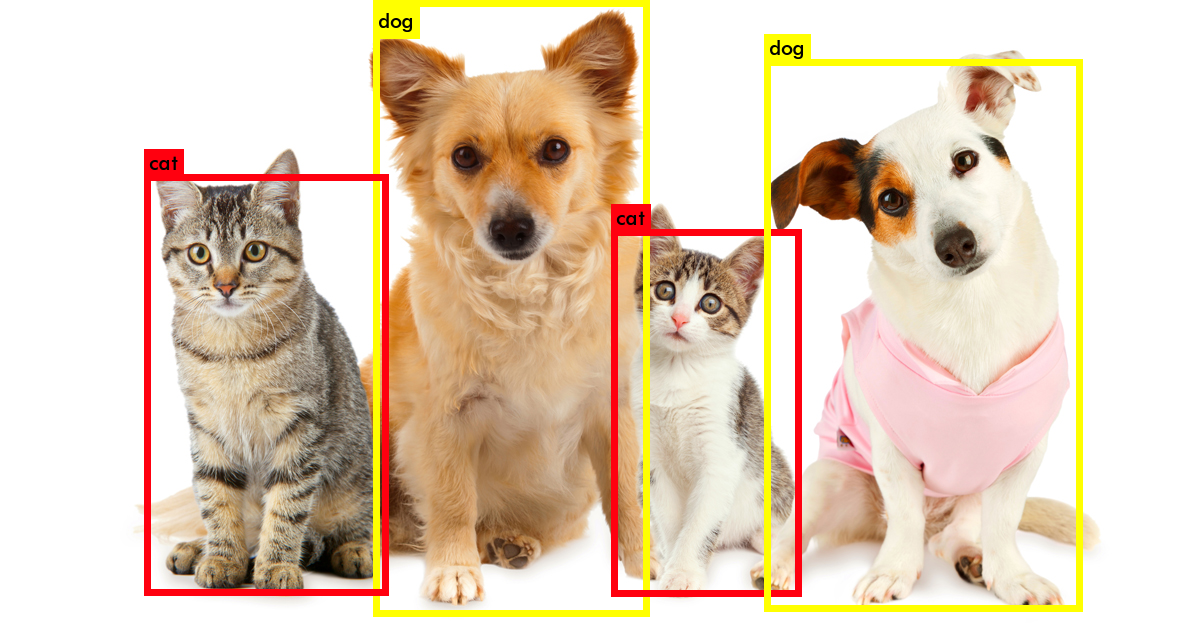
Đầu tiên sẽ tìm một hình ảnh cần predict, ở đây em sử dụng hình ảnh như bên dưới



Sau khi đã có được hình ảnh, ta sẽ upload hình ảnh này vào thư mục data của project darknet (YOLO). Sau đó thực hiện việc detect bằng câu lệnh

./darknet detect cfg/yolo.cfg yolo.weights data/dog\_cat.jpg

Sau khi detect xong kết quả trả về sẽ là một hình ảnh với các đối tượng đã được xác dịnh như bên dưới



**KẾT LUẬN**

Qua quá trình tìm hiểu và làm bài tập đã cho em nhiều hiểu biết quý giá về môn nhận dạng thị giác và ứng dụng. Giúp em có kiến thức nền tảng để có thể tiếp tục nghiên cứu thêm. Tuy nhiên, do kiến thức còn hạn chế và thời gian có hạn nên vẫn còn nhiều điểm thiếu sót.

*Tài liệu tham khảo chính:*

1. TS. Lê Đình Duy, TS Nguyễn Tấn Trần Minh Khang. Bài giảng Nhận dạng thị giác và ứng dụng.
2. Ngyễn Xuân Khánh’s Blog. <http://khanhxnguyen.com/>
3. Shelhamer, Donahue, Long. DIY Deep Learning for Vision: A Full-Day Caffe Tutorial
4. <https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm>
5. <https://en.wikipedia.org/wiki/Pattern_recognition>
6. <https://pjreddie.com/darknet/yolo/>
7. Và một số nguồn trên internet.