

MỤC LỤC

BẢNG MÔ TẢ CÁC THUẬT NGỮ	2
DANH MỤC CÁC HÌNH	3
DANH MỤC BẢNG	4
TÓM TẮT TIỂU LUẬN ĐỒ ÁN	5
Chương 1. Tổng quan	5
1. Tổng quan về đề tài	5
2. Mục tiêu, phạm vi nghiên cứu	6
3. Mô tả bài toán	6
Chương 2. Cơ sở lý thuyết	6
1. Tổng quan về Machine Learning	8
2. Thuật toán	12
2.1. Logistic Regression	12
2.2. Support Vector Machine (SVM)	15
2.3. Multi-Layer Perceptron (MLP)	23
3. Phương pháp Histogram of Oriented Gradients (HOG)	34
3.1. Khái niệm	34
3.2. Nguyên lý hoạt động	34
3.3. Các bước thực thi quy trình trích xuất đặc trưng HOG	35
3.4. Ví dụ giải tay	41
Chương 3. Xây dựng bộ dữ liệu	48
1. Cách thu thập dữ liệu (Preprocess_data_to_test)	49
2. Phân chia dữ liệu (Preprocess_and_split_data.ipynb)	52
Chương 4. Xử lý dữ liệu và áp dụng các phương pháp trích xuất trung	53
1. Tiền xử lý dữ liệu (Preproces_and_split_data)	54
2. Áp dụng kỹ thuật HOG để trích xuất đặc trưng (Take_Features_by_HOG)	55
Chương 5. Cài đặt và tinh chỉnh tham số	61
1. Thực nghiệm trên Logistic Regression	61
2. Thực nghiệm trên Support vector machine (SVM)	62
3. Thực nghiệm trên Multi layer Perceptron (MLPClassifier)	63
Chương 6. Training và đánh giá kết quả	63
1. Kết quả với HOG	64

2. Nhận xét chung	65
Chương 7. Ứng dụng và cải thiện	66
1. Ứng dụng	66
2. Hướng cải thiện	67
Danh mục tài liệu tham khảo	68
SOURCE	68

BẢNG MÔ TẢ CÁC THUẬT NGỮ

STT	Thuật ngữ tiếng Anh	Thuật ngữ tiếng Việt
1	Accuracy	Độ chính xác
2	Activation Function	Hàm kích hoạt
3	CNN (Convolutional Neural Network)	Mạng nơ-ron tích chập
4	Confusion Matrix	Ma trận nhầm lẫn
5	Cross-Entropy Loss	Hàm mất mát Entropy chéo
6	Deep Neural Network	Mạng nơ-ron sâu
7	F1-Score	Điểm F1
8	Feature Descriptor	Mô tả đặc trưng
9	Feature Extraction	Trích xuất đặc trưng
10	Feature Vector	Vector đặc trưng
11	Feed-forward Neural Network	Mạng nơ-ron truyền thẳng
12	Gamma	Tham số Gamma
13	Gradient	Độ dốc
14	Gradient Descent	Hạ Gradient
15	Gradient Magnitude	Độ lớn Gradient
16	Gradient Orientation	Phương Gradient
17	HOG Descriptor	Bộ mô tả HOG
18	Histogram	Biểu đồ tần suất
19	Histogram of Oriented Gradients (HOG)	Phương pháp mô tả đặc trưng
20	Hyperplane	Siêu phẳng
21	Kernel	Nhân
22	L2 Norm	Chuẩn L2
23	Label Encoding	Mã hóa nhãn
24	Linear Interpolation	Nội suy tuyến tính
25	Logistic Regression	Hồi quy Logistic

26	Loss Function	Hàm mất mát
27	Machine Learning (ML)	Học máy
28	Margin	Lề
29	Multi-Layer Perceptron (MLP)	Mạng nơ-ron nhiều lớp
30	Normalization	Chuẩn hóa
31	One-hot Encoding	Mã hóa One-hot
32	Outliers	Giá trị ngoại lai
33	Overfitting	Quá khớp
34	Precision	Độ chính xác dự đoán đúng
35	Preprocessing	Tiền xử lý dữ liệu
36	RBF Kernel	Nhân hàm cơ sở xuyên tâm
37	Recall	Độ nhạy
38	ReLU (Rectified Linear Unit)	Hàm ReLU
39	Regularization	Điều chuẩn
40	Sigmoid Function	Hàm Sigmoid
41	Soft Margin	Biên mềm
42	Softmax Function	Hàm Softmax
43	Sobel Filter	Bộ lọc Sobel
44	Standardization	Tiêu chuẩn hóa
45	Step Size	Kích thước bước
46	Supervised Learning	Học máy có giám sát
47	Support Vector Machine (SVM)	Máy vector hỗ trợ
48	Tanh Function	Hàm Tanh
49	Test Set	Tập kiểm tra
50	Training Set	Tập huấn luyện
51	Validation Set	Tập kiểm định

DANH MỤC CÁC HÌNH

Hình 1. Mô phỏng bài toán.....	6
Hình 2. Mối quan hệ giữa AI, Machine Learning và Deep Learning.	9
Hình 3.Học máy có giám sát (Supervised Learning)	10
Hình 4. Lý thuyết hoạt động của ML.....	11
Hình 5. Đồ thị biểu diễn kết quả ví dụ giải tay Logistic Regression	15
Hình 6. Bài toán tối ưu Lagrange bằng CVXOPT	18
Hình 7. Đồ thị kết quả bài toán SVM	21
Hình 8.Phân tích bài toán SVM.....	22
Hình 9.Các điểm gần mặt phân cách nhất của hai classes được khoanh tròn	23
Hình 10.Hàm sigmoid (trái) và tanh (phải).....	24
Hình 11. Hàm ReLU và tốc độ hội tụ khi so sánh với hàm tanh.	24
Hình 12.MLP với hai hidden layers (các biases đã bị ẩn).	25

Hình 13. Các ký hiệu sử dụng trong MLP.....	26
Hình 14. Tập dữ liệu 3 lớp theo hình xoắn ốc	28
Hình 15. Kết quả hiển thị 3 vùng màu khác nhau cho 3 classes	31
Hình 16. Kết quả với số lượng units trong hidden layer là khác nhau.....	32
Hình 17. Mạng MLP giải quyết bài toán XOR.....	33
Hình 18. Kết quả bài toán XOR.....	34
Hình 19. Kết quả quy trình trích xuất đặc trưng của HOG	40
Hình 20. Mapping độ lớn gradients với các bins	44
Hình 21. Biểu đồ Histogram of Gradient gồm 9 bins tương ứng với một ô vuông trong lưới ô vuông.....	46
Hình 22. Mẫu dữ liệu đã được thu thập	50
Hình 23. Phiếu mẫu để thu thập dữ liệu.....	50
Hình 24. PieChart	53

DANH MỤC BẢNG

Bảng 1. Kết quả ví dụ LR	15
--------------------------------	----

TÓM TẮT TIỂU LUẬN ĐỒ ÁN

Đồ án tập trung vào bài toán phân loại chữ viết tay tiếng Việt có dấu – một vấn đề quan trọng nhưng chưa được khai thác sâu trong lĩnh vực nhận dạng ký tự. Hệ thống sử dụng các kỹ thuật trích xuất đặc trưng Histogram of Oriented Gradients (HOG) và áp dụng các mô hình học máy gồm Logistic Regression, Support Vector Machine (SVM), và Multi-Layer Perceptron (MLP)

Nội dung tiểu luận bao gồm 7 chương:

Chương I. Tổng quan

Chương II. Cơ sở lý thuyết

Chương III. Xây dựng bộ dữ liệu


Chương IV. Xử lý dữ liệu và áp dụng các phương pháp trích xuất trưng

Chương V. Cài đặt và tinh chỉnh tham số

Chương VI. Training và đánh giá kết quả

Chương VII. Ứng dụng và cải thiện

Chương 1. Tổng quan

 Nội dung chương này sẽ trình bày tổng quan về bài toán nhận dạng chữ viết tay, bao gồm thực trạng nghiên cứu hiện nay, mục tiêu và phạm vi nghiên cứu, cùng mô tả chi tiết bài toán được thực hiện

1. Tổng quan về đề tài

Đến thời điểm này, trên thế giới cũng như ở Việt Nam, bài toán nhận dạng chữ viết tay vẫn còn là vấn đề thách thức lớn đối với các nhà nghiên cứu. Từ những năm 1990 đến nay, các hệ thống nhận dạng thời gian thực được xây dựng và phát triển trên cơ sở các phương pháp luận phân lớp trong lĩnh vực học máy kết hợp với các kỹ thuật xử lý ảnh một cách hiệu quả. Một số phương pháp học máy tiên tiến như mạng nơ ron, mô hình Markov ẩn, SVM,... đã được các nhà nghiên cứu trong và ngoài nước áp dụng để phát triển các ứng dụng trong lĩnh vực nhận dạng chữ.

Bài toán nhận diện chữ cái viết tay mang lại một lợi ích rất lớn trong các hoạt động thường ngày. Nhưng trước khi giải quyết được bài toán lớn đó thì phải giải quyết được một bài toán nhỏ khác đó là phân loại chữ cái viết tay. Mặc dù trong nước đã có nhiều kết quả nghiên cứu về nhận dạng chữ viết tay, tuy nhiên

các kết quả chủ yếu tập trung vào việc nhận dạng chữ số và chữ cái hệ Latinh, rất ít công trình nghiên cứu đề xuất các giải pháp cho việc nhận dạng chữ viết tay tiếng Việt.

2. Mục tiêu, phạm vi nghiên cứu

- Nghiên cứu và phát triển các phương pháp trích chọn đặc trưng nhằm tăng khả năng phân biệt giữa các ký tự tiếng Việt có dấu.
- Đánh giá ảnh hưởng của các đặc trưng này đến hiệu suất phân loại.
- Thử nghiệm hệ thống với các tập dữ liệu thực tế để đánh giá độ chính xác, hiệu suất, và khả năng ứng dụng.

3. Mô tả bài toán

- Bài toán này thuộc lớp bài toán phân loại, có tổng cộng 89 lớp đại diện cho 89 chữ cái tiếng Việt viết thường bao gồm cả các dấu phụ (sắc, huyền, hỏi, ngã, nặng). Đầu vào của bài toán là một tấm ảnh trong đó có chứa đúng một chữ cái tiếng Việt viết thường.
- Đầu ra là kết quả dự đoán chữ cái tương ứng với tấm ảnh đó.



Hình 1. Mô phỏng bài toán

Chương 2. Cơ sở lý thuyết

👉 Mục tiêu chính là tạo nền tảng lý thuyết và kỹ thuật cho việc xây dựng và triển khai các mô hình học máy. Tập trung vào Supervised Learning, chương trình bày hai loại bài toán chính: Classification và Regression, cùng các thuật toán như Logistic Regression, SVM, và MLP. Giới thiệu phương pháp HOG để trích xuất đặc trưng hình ảnh, làm nền tảng lý thuyết và kỹ thuật cho việc triển khai mô hình phân loại chữ viết tay tiếng Việt.

❖ Tổng quan về Machine Learning phân loại các nhóm học máy

- Gồm 4 nhóm chính: Supervised, Unsupervised, Semi-supervised, và Reinforcement learning.
- Đồ án thuộc nhóm Supervised Learning, sử dụng dữ liệu đã gán nhãn để huấn luyện mô hình.

❖ Thuật toán học máy được sử dụng

- Logistic Regression: Mô hình phân loại nhị phân dựa trên hàm sigmoid.
 - Khái niệm: Thuật toán học máy có giám sát, sử dụng hàm logistic để phân loại dữ liệu thành hai hoặc nhiều nhóm.
 - Nguyên lý hoạt động: Tính toán hàm tuyến tính, chuyển đổi qua hàm sigmoid để ước lượng xác suất, phân loại dựa trên xác suất và tối ưu hóa tham số bằng hàm mất mát (Log-Loss).
 - Ví dụ giải tay: Minh họa mối quan hệ giữa thời gian ôn thi và xác suất đậu kỳ thi.
 - Cách thức thực thi:
 - Tính hàm tuyến tính từ dữ liệu.
 - Chuyển đổi qua hàm sigmoid.
 - Phân loại dựa trên ngưỡng xác suất (thường là 0.5).
 - Tối ưu tham số bằng thuật toán Gradient Descent.
- SVM: Tìm siêu phẳng tối ưu để phân tách các lớp dữ liệu.
 - Khái niệm: Thuật toán học máy có giám sát, tìm siêu phẳng tối ưu để phân tách các lớp dữ liệu khác nhau.
 - Nguyên lý hoạt động: Tối đa hóa margin, sử dụng kernel để xử lý dữ liệu phi tuyến, và tối ưu hóa thông qua hàm mục tiêu.
 - Cách thức thực thi:
 - Thiết lập bài toán tối ưu bằng cách xác định siêu phẳng với margin lớn nhất.
 - Sử dụng kernel để xử lý dữ liệu phi tuyến.
 - Giải bài toán tối ưu bằng phương pháp Lagrange (dạng đối ngẫu).
 - Dự đoán điểm dữ liệu mới dựa trên vị trí so với siêu phẳng.
 - Ví dụ giải tay: Minh họa việc tìm siêu phẳng phân tách tối ưu và các vector hỗ trợ.
- MLP: Mạng nơ-ron truyền thẳng với khả năng xử lý dữ liệu phức tạp.
 - Khái niệm: Mạng nơ-ron truyền thẳng gồm các lớp đầu vào, ẩn, và đầu ra.

- Nguyên lý hoạt động: Thông tin truyền từ input qua các hidden layers với các hàm kích hoạt (ReLU, Sigmoid, Tanh) đến output.
- Cách thức thực thi:
 - Xây dựng cấu trúc mạng với các lớp input, hidden, và output.
 - Áp dụng các hàm kích hoạt trong từng lớp.
 - Tối ưu mô hình bằng cách giảm thiểu hàm mất mát thông qua thuật toán Backpropagation và Gradient Descent.
- Ví dụ giải tay: Giải quyết bài toán XOR và minh họa khả năng phân loại phi tuyến.

➤ Phương pháp HOG

- Khái niệm: Phương pháp trích xuất đặc trưng từ hình ảnh dựa trên gradient và hướng.
- Nguyên lý hoạt động: Chia hình ảnh thành các ô vuông, tính histogram gradient cho từng ô, chuẩn hóa theo block, và ghép các vector để tạo thành đặc trưng HOG.
- Cách thức thực thi:
 - Tiền xử lý: Chuyển ảnh sang grayscale, chỉnh kích thước.
 - Tính gradient bằng bộ lọc Sobel.
 - Chia ảnh thành các ô (cells) và tính histogram hướng gradient cho từng ô.
 - Chuẩn hóa vector histogram trong các block (gồm 2x2 ô).
 - Kết hợp các vector để tạo thành đặc trưng HOG cho toàn bộ ảnh.
- Ví dụ giải tay: Minh họa việc tính toán HOG cho hình ảnh và biểu diễn đặc trưng gradient qua histogram.

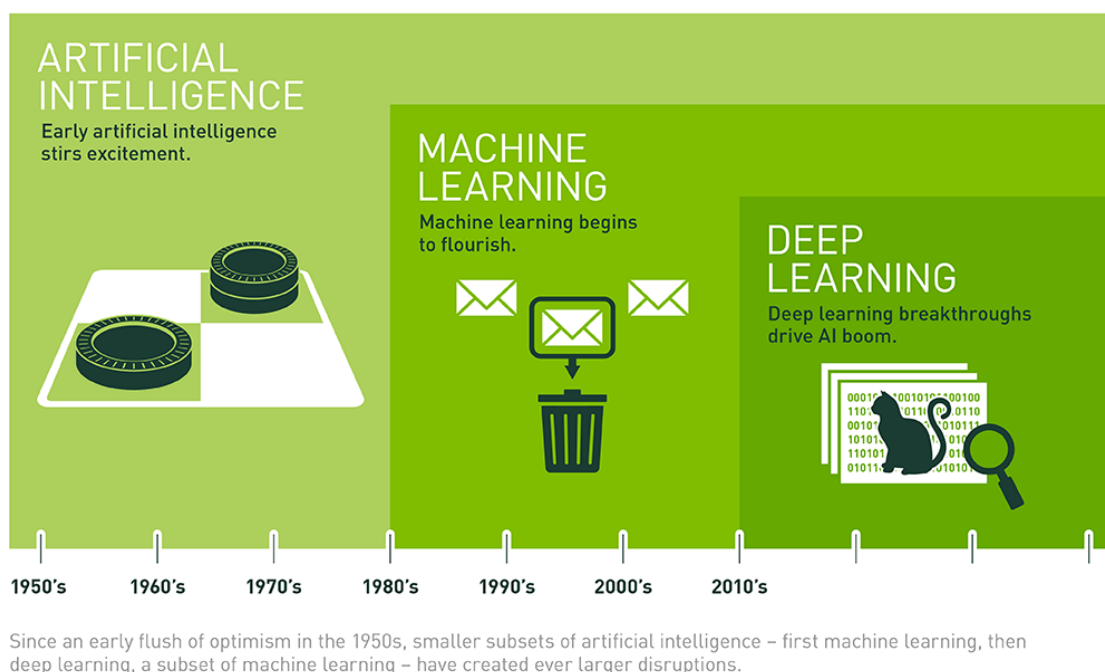
1. Tổng quan về Machine Learning

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, ..., chỉ là một vài trong vô vàn những ứng dụng của AI/ Machine Learning.

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, Machine learning is the subfield of computer science that “gives computers the ability to learn without being explicitly

programmed”. Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể

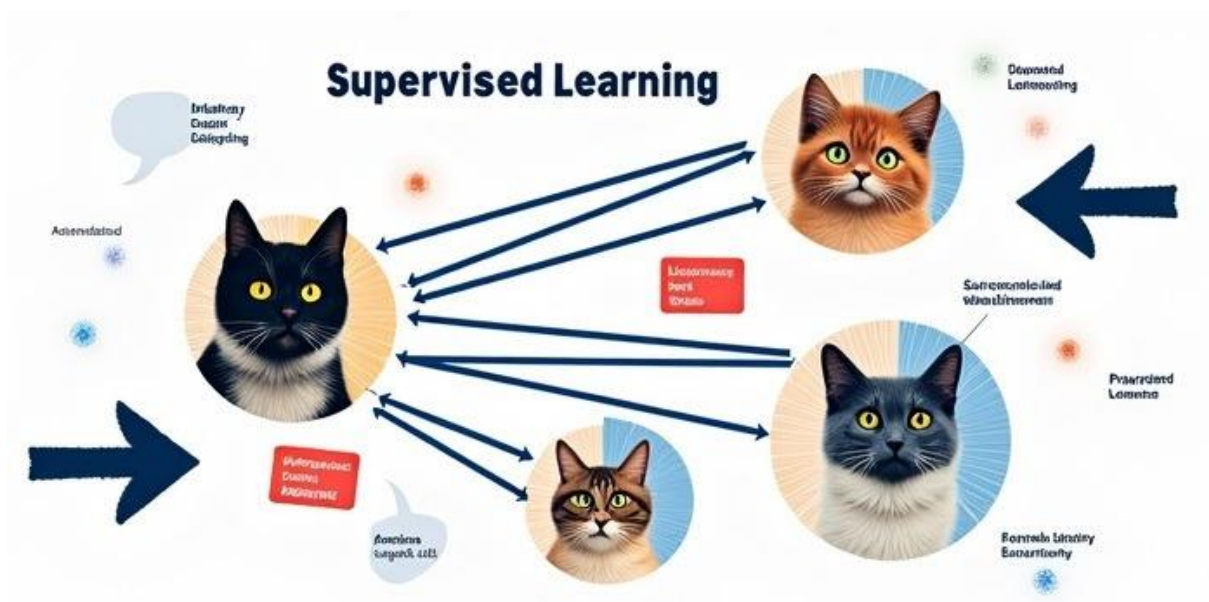
Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học sâu). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc



Hình 2. Mối quan hệ giữa AI, Machine Learning và Deep Learning.

(Nguồn: [What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?](#))

Theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Supervised learning, Unsupervised learning, Semi-supervised learning và Reinforcement learning. Có một số cách phân nhóm không có Semi-supervised learning hoặc Reinforcement learning. Tuy nhiên, đề án “Phân loại chữ viết tay tiếng Việt có dấu” thuộc Supervised learning, cụ thể như sau:

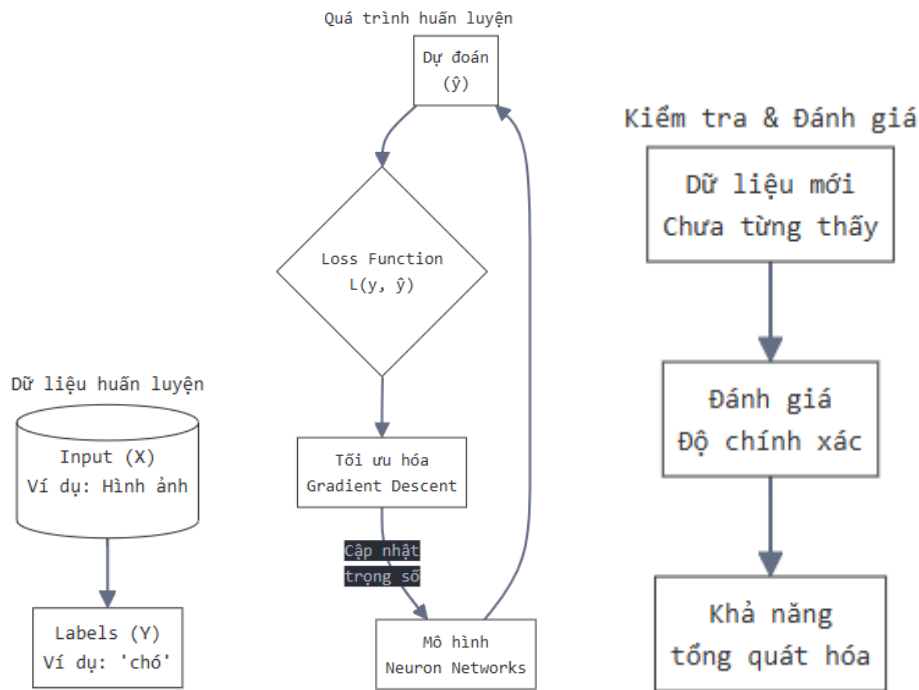


Hình 3. Học máy có giám sát (Supervised Learning)

Định nghĩa:

- Học máy có giám sát là một phương pháp trong học máy, trong đó mô hình được huấn luyện trên một tập dữ liệu đã được gán nhãn trước. Mỗi mẫu dữ liệu đầu vào (input) được liên kết với một nhãn đầu ra (output) mong muốn. Mục tiêu của mô hình là học cách ánh xạ từ đầu vào đến đầu ra dựa trên các cặp dữ liệu (Input, Output) này.
- Thuật toán học có giám sát có nhiệm vụ dự đoán đầu ra (Outcome) của một dữ liệu mới (New input) dựa trên kiến thức học được từ các cặp dữ liệu đã biết trước đó. Các cặp dữ liệu này thường được gọi là Data/ Label, tức dữ liệu/nhãn.

Nguyên lý hoạt động:



Hình 4. Lý thuyết hoạt động của ML

- **Dữ liệu huấn luyện:** Bao gồm các cặp dữ liệu đầu vào và đầu ra (ví dụ: hình ảnh của một con chó và nhãn “chó”).
- **Quá trình huấn luyện:** Mô hình học cách dự đoán đầu ra từ đầu vào bằng cách tối ưu hóa một hàm mất mát (loss function), thường là sự khác biệt giữa dự đoán của mô hình và nhãn thực tế.
- **Kiểm tra:** Sau khi huấn luyện, mô hình được kiểm tra trên dữ liệu chưa từng thấy để đánh giá độ chính xác và khả năng tổng quát hóa.

Thuật toán supervised learning còn được tiếp tục chia nhỏ ra thành hai loại chính:

- **Classification (Phân loại):** Một bài toán được gọi là classification nếu các label của input data được chia thành một số hữu hạn nhóm. Ví dụ: Gmail xác định xem một email có phải là spam hay không; các hãng tín dụng xác định xem một khách hàng có khả năng thanh toán nợ hay không. Ba ví dụ phía trên được chia vào loại này.
- **Regression (Hồi quy):** Nếu label không được chia thành các nhóm mà là một giá trị thực cụ thể. Ví dụ: một căn nhà rộng x m², có y phòng ngủ và cách trung tâm thành phố z km sẽ có giá là bao nhiêu?

Ứng dụng phổ biến

- **Phân loại (Classification):** phân loại email là spam hay không,...
- **Hồi quy (Regression):** Dự đoán giá nhà dựa trên các đặc điểm như diện tích, vị trí, v.v.

2. Thuật toán

2.1. Logistic Regression

2.1.1. Khái niệm

Hồi quy logistic là một phương pháp học máy có giám sát được sử dụng để phân loại dữ liệu thành hai hoặc nhiều nhóm. Phương pháp này ước lượng xác suất xảy ra của một sự kiện dựa trên các biến đầu vào, bằng cách sử dụng hàm logistic để ánh xạ bất kỳ giá trị thực nào thành khoảng (0, 1). Ví dụ: xác suất thi đỗ nếu biết thời gian ôn thi, xác suất ngày mai có mưa dựa trên những thông tin đo được trong ngày hôm nay,...

2.1.2. Nguyên lý hoạt động

- **Tính toán hàm tuyến tính:**

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

- **Chuyển đổi qua hàm sigmoid** để đưa z về khoảng xác suất [0,1]:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Phân loại** dựa trên xác suất:

- $y = 1$ nếu $\sigma(z) \geq 0.5$.
- $y = 0$ nếu $\sigma(z) < 0.5$.

- **Tối ưu hóa tham số β** bằng cách giảm thiểu hàm mất mát (**Log-Loss**):

$$\text{Loss} = -\frac{1}{m} \sum_{i=1}^m [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

- **Sử dụng Gradient Descent, HOG** hoặc các thuật toán tối ưu khác để huấn luyện mô hình.

2.1.3. Cách thức thực thi

Tính toán hàm tuyến tính

```
[3] import numpy as np

# X: Ma trận đặc trưng (bao gồm cột bias)
# beta: Trọng số ban đầu
def compute_linear(X, beta):
    return np.dot(X, beta)
```

Chuyển đổi qua hàm sigmoid

```
[4] def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Phân loại dựa trên xác suất

```
[5] def classify(y_prob, threshold=0.5):
    return (y_prob >= threshold).astype(int)
```

Tối ưu hóa tham số β bằng cách giảm thiểu hàm mất mát (Log-Loss)

```
[6] def compute_loss(y_true, y_pred):
    m = len(y_true)
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss
```

Sử dụng Gradient Descent để huấn luyện mô hình

```
[7] def gradient_descent(X, y, beta, learning_rate, epochs):
    m = len(y)
    for epoch in range(epochs):
        # Tính toán dự đoán
        z = compute_linear(X, beta)
        y_pred = sigmoid(z)

        # Tính gradient
        gradient = np.dot(X.T, (y_pred - y)) / m

        # Cập nhật trọng số
        beta -= learning_rate * gradient

        # In thông tin (tùy chọn)
        if epoch % 100 == 0:
            loss = compute_loss(y, y_pred)
            print(f"Epoch {epoch}, Loss: {loss:.4f}")
    return beta
```

Triển khai Logistic Regression hoàn chỉnh

```
[8] # Khởi tạo dữ liệu
X = np.array([[1, 2], [1, 3], [1, 5]]) # Dữ liệu với cột bias
y = np.array([0, 0, 1]) # Nhãn
beta = np.zeros(X.shape[1]) # Khởi tạo trọng số

# Huấn luyện mô hình
learning_rate = 0.01
epochs = 1000
beta = gradient_descent(X, y, beta, learning_rate, epochs)

# Dự đoán
z = compute_linear(X, beta)
y_pred = sigmoid(z)
y_classified = classify(y_pred)
print("Dự đoán phân loại:", y_classified)
```

```
↔ Epoch 0, Loss: 0.6931
Epoch 100, Loss: 0.6677
Epoch 200, Loss: 0.6438
Epoch 300, Loss: 0.6213
Epoch 400, Loss: 0.6001
Epoch 500, Loss: 0.5800
Epoch 600, Loss: 0.5610
Epoch 700, Loss: 0.5431
Epoch 800, Loss: 0.5261
Epoch 900, Loss: 0.5100
Dự đoán phân loại: [0 0 1]
```

2.1.4. Ví dụ giải tay

Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi. Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào?

Kết quả thu được như sau:

Số giờ	Đậu	Số giờ	Đậu
0.5	0	2.75	1
0.75	0	3	0
1	0	3.25	1
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1

2.25	1	5	1
2.5	0	5.5	1

Bảng 1. Kết quả ví dụ LR

Triển khai các hàm cần thiết cho logistic sigmoid regression:

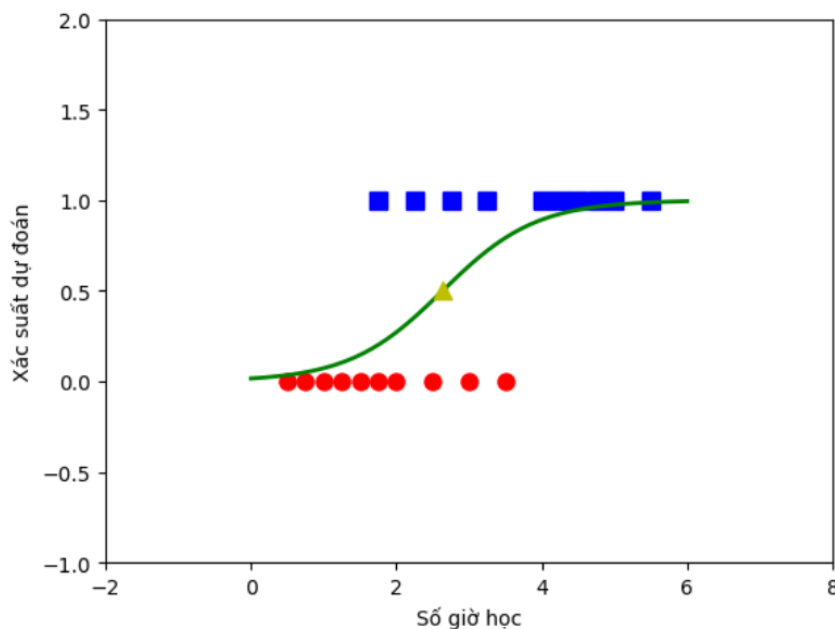
⇒ `[[-4.092695]`
 => Được kết quả: `[1.55277242]]`

Với kết quả tìm được, đầu ra y có thể được dự đoán theo công thức: $y = \text{sigmoid}(-4.1 + 1.55 \cdot x)$. Với dữ liệu trong tập training, kết quả là:

```
[11] print(sigmoid(np.dot(w[-1].T, X)))
```

```
⇒ [[0.03501592 0.05078108 0.07310642 0.10416972 0.14634799 0.20175793
    0.20175793 0.27147534 0.35458234 0.4475032 0.54424128 0.63775242
    0.72188183 0.79282004 0.89267823 0.92460236 0.94758783 0.96384008
    0.97518471 0.9884283 ]]
```

Biểu diễn kết quả này trên đồ thị ta có:



Hình 5. Đồ thị biểu diễn kết quả ví dụ giải tay Logistic Regression

Logistic Regression học được mối quan hệ giữa số giờ học và xác suất đậu kỳ thi. Đường cong sigmoid cho thấy mô hình chuyển từ vùng $y=0$ (rớt) sang vùng $y=1$ (đậu) dựa trên một ngưỡng, thường là 0.5

2.2. Support Vector Machine (SVM)

2.2.1. Khái niệm

Support Vector Machine (SVM) là một thuật toán học máy có giám sát, chủ yếu được sử dụng cho các tác vụ phân loại và hồi quy. SVM dựa trên lý thuyết học thống kê do Vapnik và Chervonenkis (1995) xây dựng. Mục tiêu chính của SVM là tìm ra một siêu phẳng tối ưu để phân tách các lớp dữ liệu khác nhau trong không gian nhiều chiều. Siêu phẳng này được xác định sao cho khoảng cách (margin) đến các điểm dữ liệu gần nhất của mỗi lớp, gọi là các vector hỗ trợ (support vectors), là lớn nhất, nhằm tăng cường khả năng phân loại chính xác và khả năng tổng quát hóa của mô hình.

2.2.2. Nguyên lý hoạt động

Tìm siêu phẳng tối ưu: SVM cố gắng tìm siêu phẳng có khoảng cách lớn nhất (margin) đến các điểm dữ liệu gần nhất từ cả hai lớp, được gọi là các vector hỗ trợ (support vectors). Margin càng lớn, mô hình càng tổng quát.

Làm việc trong không gian cao hơn: Nếu dữ liệu không thể phân tách tuyến tính trong không gian ban đầu, SVM sử dụng **hàm kernel** để ánh xạ dữ liệu sang một không gian cao chiều, nơi nó có thể phân tách tuyến tính.

Định nghĩa hàm mục tiêu: SVM tối ưu hóa một hàm mục tiêu để tìm siêu phẳng với margin tối đa, đồng thời giảm thiểu lỗi phân loại thông qua một tham số điều chỉnh CCC, giúp cân bằng giữa độ chính xác trên tập huấn luyện và khả năng tổng quát.

Dự đoán: Sau khi siêu phẳng được xác định, SVM sử dụng nó để phân loại các điểm dữ liệu mới dựa trên vị trí của chúng so với siêu phẳng.

2.2.2.1 Thiết lập bài toán tối ưu:

- **Dữ liệu đầu vào:**
 - Một tập dữ liệu huấn luyện $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, với:
 - x_i : Đặc trưng của mẫu dữ liệu.
 - $y_i \in \{-1, 1\}$: Nhãn lớp của dữ liệu.
- **Hàm mục tiêu:**
 - Tìm siêu phẳng $w^T x + b = 0$ sao cho biên giữa hai lớp là lớn nhất.
 - Bài toán tối ưu:
$$\min_{w, b} \frac{1}{2} \|w\|^2$$
 - Với ràng buộc:
$$y_i(w^T x_i + b) \geq 1, \forall i$$
- **Soft Margin SVM:**

- Thêm biến $\xi_i \geq 0$ để cho phép lỗi phân loại.

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

- Bài toán tối ưu trở thành:
- Với C: Tham số điều chỉnh giữa lỗi phân loại và độ phức tạp của mô hình.

2.2.2.2 Giải bài toán tối ưu:

SVM giải bài toán tối ưu thông qua hai dạng:

- **Dạng nguyên bản (Primal):** Tối ưu trực tiếp hàm mục tiêu trong không gian đầu vào.
 - **Dạng đối ngẫu (Dual):**
 - Áp dụng phương pháp Lagrange để chuyển bài toán thành dạng đối ngẫu, nhằm tối ưu hiệu quả hơn khi dữ liệu lớn hoặc khi sử dụng kernel.
- $$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$
- Hàm mục tiêu đối ngẫu: Với $K(x_i, x_j)$: Hàm kernel.

2.2.2.3. Sử dụng kernel để xử lý dữ liệu phi tuyến:

Khi dữ liệu không thể tách tuyến tính trong không gian ban đầu, kernel được sử dụng để ánh xạ dữ liệu vào không gian đặc trưng cao hơn. Các kernel phổ biến:

- Linear Kernel: $K(x_i, x_j) = x_i^T x_j$
- Polynomial Kernel: $K(x_i, x_j) = (x_i^T x_j + c)^d$
- RBF Kernel (Gaussian): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$

2.2.2.4. Dự đoán trên dữ liệu mới:

Sau khi huấn luyện, siêu phẳng phân tách được xác định. Dự đoán cho một điểm mới xx được thực hiện dựa trên dấu của:

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b$$

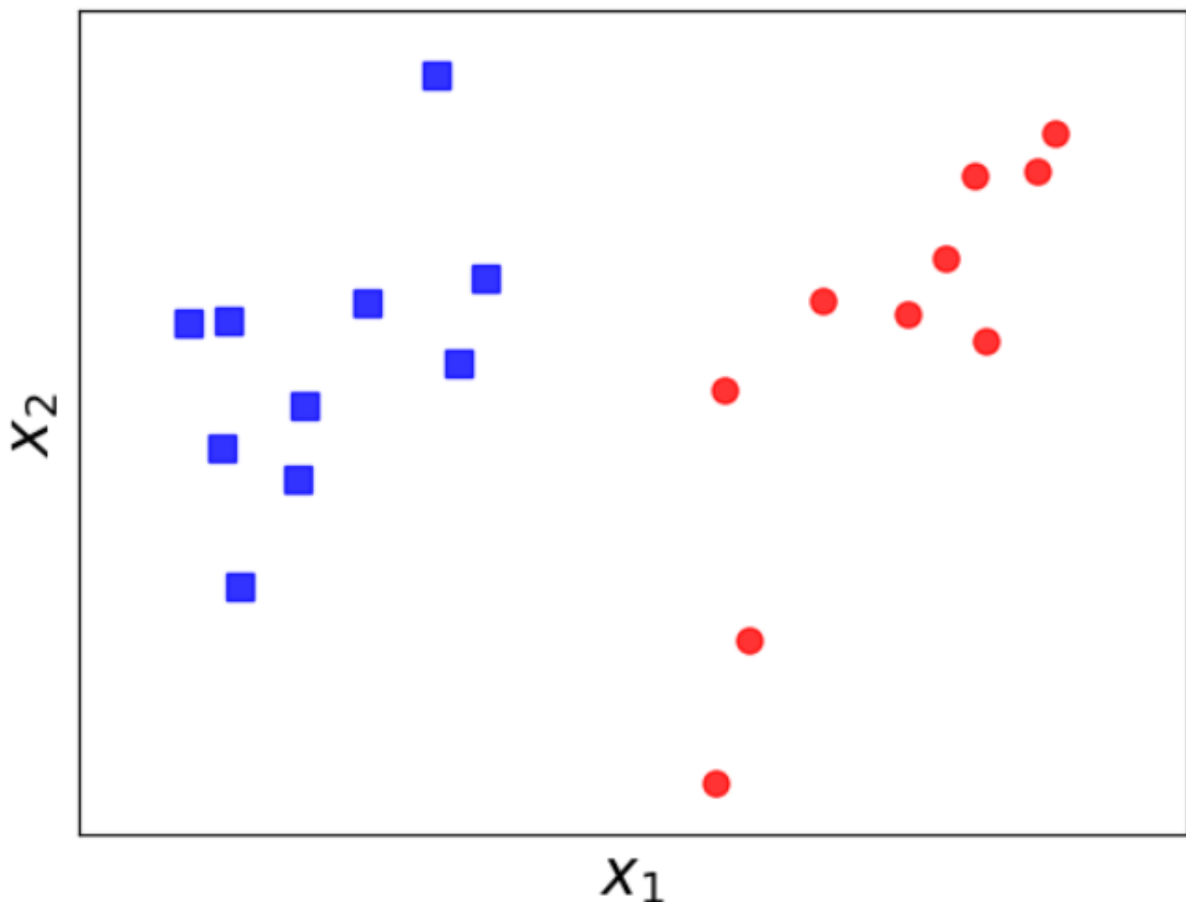
- Nếu $f(x) > 0$: Thuộc lớp +1
- Nếu $f(x) < 0$: Thuộc lớp -1

2.2.3. Cách thức thực thi

Giải bài toán bài toán đối ngẫu Lagrange bằng CVXOPT

$$\begin{aligned} \lambda &= \arg \max_{\lambda} g(\lambda) \\ \text{subject to: } \lambda &\succeq 0 \quad (9) \\ \sum_{n=1}^N \lambda_n y_n &= 0 \end{aligned}$$

Bài toán tạo một tập dữ liệu mẫu cho thuật toán Support Vector Machine (SVM) trong không gian hai chiều, bao gồm hai lớp dữ liệu (lớp 1 và lớp -1), mỗi lớp có 10 điểm được tạo ra theo phân phối chuẩn đa biến. Lớp 1 được tập trung quanh tọa độ (2,2) và lớp -1 tập trung quanh tọa độ (4,2), với độ phân tán được kiểm soát bởi một ma trận hiệp phương sai định trước. Mục tiêu là tạo ra một tập dữ liệu có tính chất phân tách được để có thể áp dụng thuật toán SVM, từ đó tìm ra đường phân cách tối ưu và các support vectors, cuối cùng là trực quan hóa kết quả phân loại thông qua biểu đồ.



Hình 6. Bài toán đối ngẫu Lagrange bằng CVXOPT

Đây là phần giải quyết bài toán tối ưu hóa bậc hai (quadratic programming) để tìm các hệ số lambda trong SVM, Lambda là các hệ số Lagrange, đóng vai trò quan trọng trong việc xác định các support vectors

```

from cvxopt import matrix, solvers
# Xây dựng ma trận K
V = np.concatenate((X0.T, -X1.T), axis = 1)
K = matrix(V.T.dot(V)) # xem định nghĩa của V, K
# Vector p là vector toàn số 1 âm
p = matrix(-np.ones((2*N, 1)))
# Xây dựng các ma trận ràng buộc A, b, G, h
G = matrix(-np.eye(2*N)) # để đảm bảo tất cả  $\lambda_n \geq 0$ 
h = matrix(np.zeros((2*N, 1)))
A = matrix(y) # ràng buộc đẳng thức là  $y^T \lambda = 0$ 
b = matrix(np.zeros((1, 1)))
# Tắt hiển thị quá trình tối ưu
solvers.options['show_progress'] = False
# Giải bài toán quy hoạch bậc hai
sol = solvers.qp(K, p, G, h, A, b)
# Lấy kết quả lambda
l = np.array(sol['x'])
print('lambda = ')
print(l.T)

```

Kết quả cho phép tìm được siêu phẳng tối ưu phân tách hai lớp. Các điểm có $\lambda > 0$ sẽ trở thành support vectors, là những điểm quan trọng nhất trong việc xác định đường biên phân tách giúp tìm ra đường biên phân tách tối ưu giữa hai lớp dữ liệu.

```

lambda =
[[8.54018321e-01 2.89132533e-10 1.37095535e+00 6.36030818e-10
 4.04317408e-10 8.82390106e-10 6.35001881e-10 5.49567576e-10
 8.33359230e-10 1.20982928e-10 6.86678649e-10 1.25039745e-10
 2.22497367e+00 4.05417905e-09 1.26763684e-10 1.99008949e-10
 2.13742578e-10 1.51537487e-10 3.75329509e-10 3.56161975e-10]]

```

Những điểm có λ lớn hơn 0 được gọi là Support Vectors (các vector hỗ trợ). Trong trường hợp này, có 3 support vectors chính các điểm còn lại (có $\lambda \approx 0$) không phải là support vectors và không ảnh hưởng đến việc xác định đường biên phân tách. Ba điểm dữ liệu có λ lớn này là những điểm quan trọng nhất trong việc xác định đường biên phân tách (decision boundary). Chúng nằm gần nhất với đường biên phân tách giữa hai lớp các điểm khác (có $\lambda \approx 0$) nằm xa đường biên và không ảnh hưởng đến việc phân loại.

```

# epsilon là một số dương rất nhỏ, lớn hơn 1e-9
epsilon = 1e-6

# Tìm các chỉ số của các support vector (những điểm có lambda > epsilon)
S = np.where(l > epsilon)[0]

# Lấy ra các vector/ma trận tương ứng với support vector
VS = V[:, S] # Ma trận V chỉ gồm các cột support vector
XS = X[:, S] # Ma trận X chỉ gồm các cột support vector
yS = y[:, S] # Vector y chỉ gồm các phần tử support vector
lS = l[S]     # Vector lambda chỉ gồm các phần tử support vector

# Tính vector trọng số w và hệ số điều chỉnh b
w = VS.dot(lS) # Tính vector trọng số w
b = np.mean(yS.T - w.T.dot(XS)) # Tính hệ số điều chỉnh b

# In kết quả
print('w = ', w.T)
print('b = ', b)

```

Chuyển từ không gian tối ưu (lambda) sang không gian quyết định (w, b). Kết quả cuối cùng là phương trình siêu phẳng: $w^T x + b = 0$. Phương trình này sẽ được sử dụng để phân loại các điểm dữ liệu mới:

- Nếu $w^T x + b > 0$: điểm thuộc lớp 1
- Nếu $w^T x + b < 0$: điểm thuộc lớp -1

Đây là bước cuối cùng trong quá trình huấn luyện SVM, cho phép có một mô hình hoàn chỉnh để thực hiện phân loại.

```

w = [[-2.00984381  0.64068336]]

```

Vector 2 chiều vì dữ liệu đầu vào có 2 đặc trưng. Giá trị $w_1 = -2.00984381$ (trọng số cho đặc trưng thứ nhất), giá trị $w_2 = 0.64068336$ (trọng số cho đặc trưng thứ hai) vector này vuông góc với đường biên phân tách và chỉ hướng phân chia giữa hai lớp. Độ lớn của các thành phần cho thấy đặc trưng thứ nhất có ảnh hưởng lớn hơn đến việc phân loại

```

b = 4.668560633868093

```

Độ lệch của siêu phẳng phân tách giá trị dương cho thấy siêu phẳng được dịch chuyển theo hướng dương

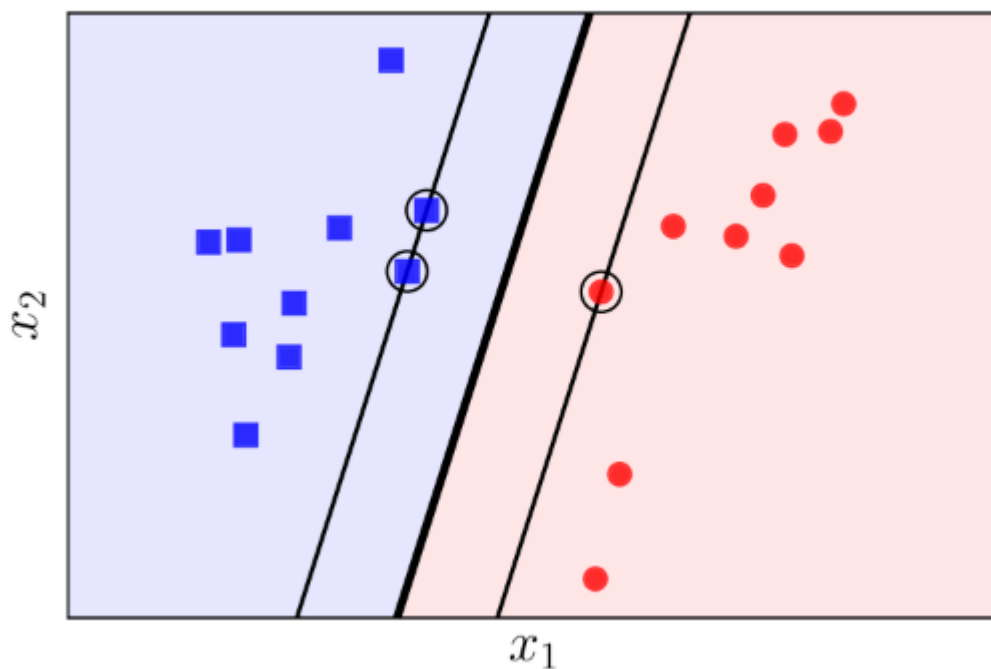
Phương trình siêu phẳng phân tách:

- $f(x) = w_1 x_1 + w_2 x_2 + b = 0$

- $f(x) = -2.00984381x_1 + 0.64068336x_2 + 4.66856063 = 0$

Cách sử dụng để phân loại:

- Với một điểm dữ liệu $x = (x_1, x_2)$:
 - Nếu $-2.00984381x_1 + 0.64068336x_2 + 4.66856063 > 0$: điểm thuộc lớp 1
 - Nếu $-2.00984381x_1 + 0.64068336x_2 + 4.66856063 < 0$: điểm thuộc lớp -1
 - Nếu $-2.00984381x_1 + 0.64068336x_2 + 4.66856063 = 0$: điểm nằm trên đường biên phân tách



Hình 7. Đồ thị kết quả bài toán SVM

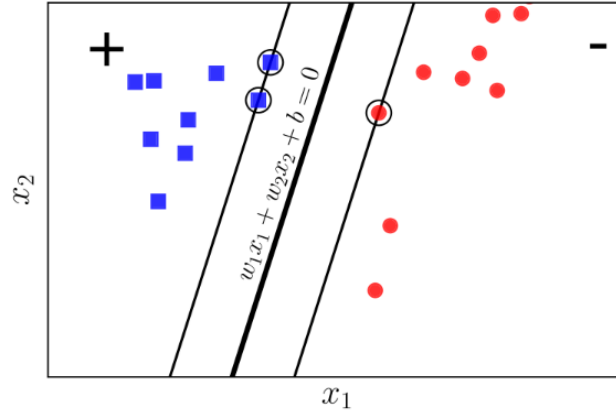
Margin rộng \rightarrow mô hình tổng quát tốt

Support vectors phù hợp \rightarrow phân loại chính xác

Phân bố điểm dữ liệu \rightarrow thấy được độ phân tách của dữ liệu

2.2.4. Ví dụ giải tay

Giả sử có một tập huấn luyện gồm các cặp dữ liệu $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$, trong đó $x_i \in \mathbb{R}^d$ là vector đầu vào của dữ liệu và y_i là nhãn của điểm dữ liệu đó. Dữ liệu có d chiều và có N điểm dữ liệu. Mỗi nhãn y_i có giá trị là 1 (class 1) hoặc -1 (class 2)



Hình 8. Phân tích bài toán SVM

Giả sử các điểm vuông xanh thuộc class 1 và các điểm tròn đỏ thuộc class -1. Mặt $w^T x + b = w_1 x_1 + w_2 x_2 + b = 0$ là mặt phân chia giữa hai lớp, trong đó class 1 nằm ở phía dương và class -1 nằm ở phía âm của mặt này. Nếu vị trí lớp bị đảo, ta chỉ cần đổi dấu của w và b .

Mục tiêu là tìm các hệ số w và b . Một điểm quan trọng được nhấn mạnh: với mỗi cặp dữ liệu (x_n, y_n) , khoảng cách từ điểm đó đến mặt phân chia có ý nghĩa quan trọng

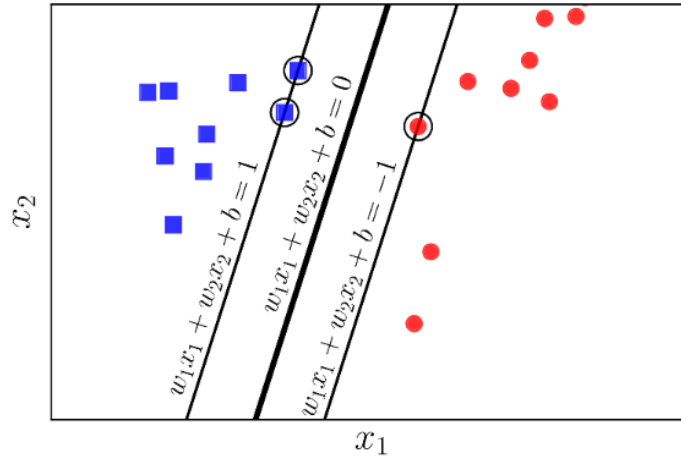
$$\frac{y_n(\mathbf{w}^T \mathbf{x}_n + b)}{\|\mathbf{w}\|_2}$$

Theo giả thiết, y_n luôn cùng dấu với phía của x_n , do đó y_n cùng dấu với $(\mathbf{w}^T \mathbf{x}_n + b)$. Điều này đảm bảo rằng tử số trong biểu thức tính khoảng cách luôn không âm. Với mặt phân chia đã cho, margin được định nghĩa là khoảng cách nhỏ nhất từ một điểm bất kỳ trong hai lớp đến mặt phân chia

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

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

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



Hình 9. Các điểm gần mặt phân cách nhất của hai classes được khoanh tròn

2.3. Multi-Layer Perceptron (MLP)

2.3.1. Khái niệm

Multi-Layer Perceptron (MLP) là một trong những dạng mạng nơ-ron nhân tạo (Artificial Neural Network – ANN) phổ biến và cơ bản nhất. MLP thuộc nhóm **mạng nơ-ron truyền thẳng (Feed-forward Neural Network)**, nghĩa là thông tin sẽ được truyền theo một chiều từ đầu vào (input) qua các lớp ẩn (hidden layers) đến đầu ra (output), không có vòng lặp hay cơ chế lan truyền ngược vòng trong kiến trúc tầng

2.3.2. Nguyên lý hoạt động

Cấu trúc mạng (Input – Hidden – Output)

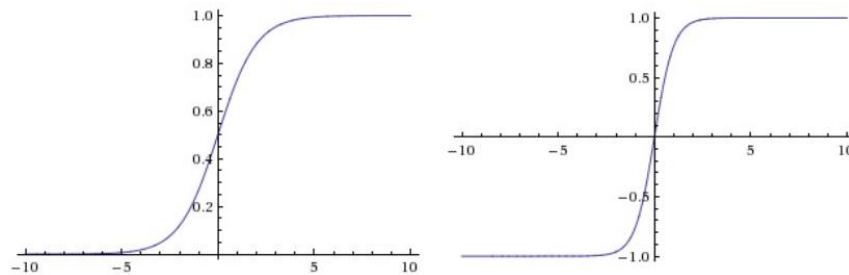
Lớp đầu vào (Input layer): Nhận dữ liệu đầu vào dưới dạng vector số (các đặc trưng của dữ liệu)

- Tiếp nhận dữ liệu hoặc đặc trưng (feature) đầu vào
- Số lượng neuron bằng số chiều của dữ liệu đầu vào

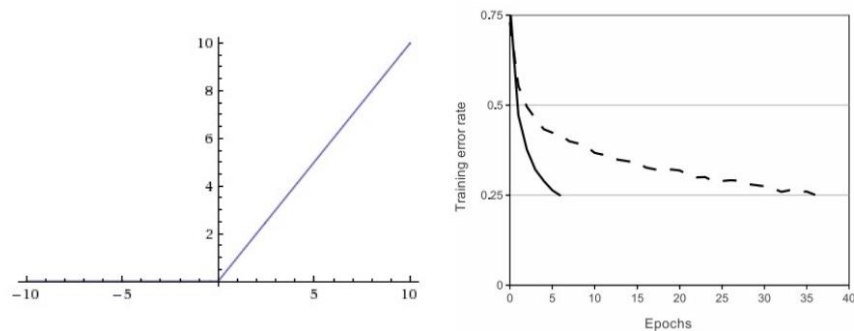
Các lớp ẩn (Hidden layers): Một hoặc nhiều lớp ẩn kết nối các neuron, chịu trách nhiệm học các mối quan hệ phức tạp

- Mỗi lớp ẩn gồm nhiều neuron (perceptron)
- Số lượng lớp ẩn có thể là 1 hoặc nhiều (Deep Neural Network)
- Mỗi neuron tính toán tổ hợp tuyến tính của đầu vào và áp dụng hàm kích hoạt (activation function)
- Các hàm kích hoạt phổ biến:

- ReLU: $f(s) = \max(0, s)$
- Sigmoid: $f(s) = 1/(1 + \exp(-s))$
- Tanh: $f(x) = (e^x - e^{-x}) / (e^x + e^{-x})$



Hình 10. Hàm sigmoid (trái) và tanh (phải)



Hình 11. Hàm ReLU và tốc độ hội tụ khi so sánh với hàm tanh.

Lớp đầu ra (Output layer): Trả về kết quả cuối cùng, có thể là phân loại, hồi quy hoặc một loại tín hiệu khác. Mỗi output của một unit (trừ các input units) được tính dựa vào công thức

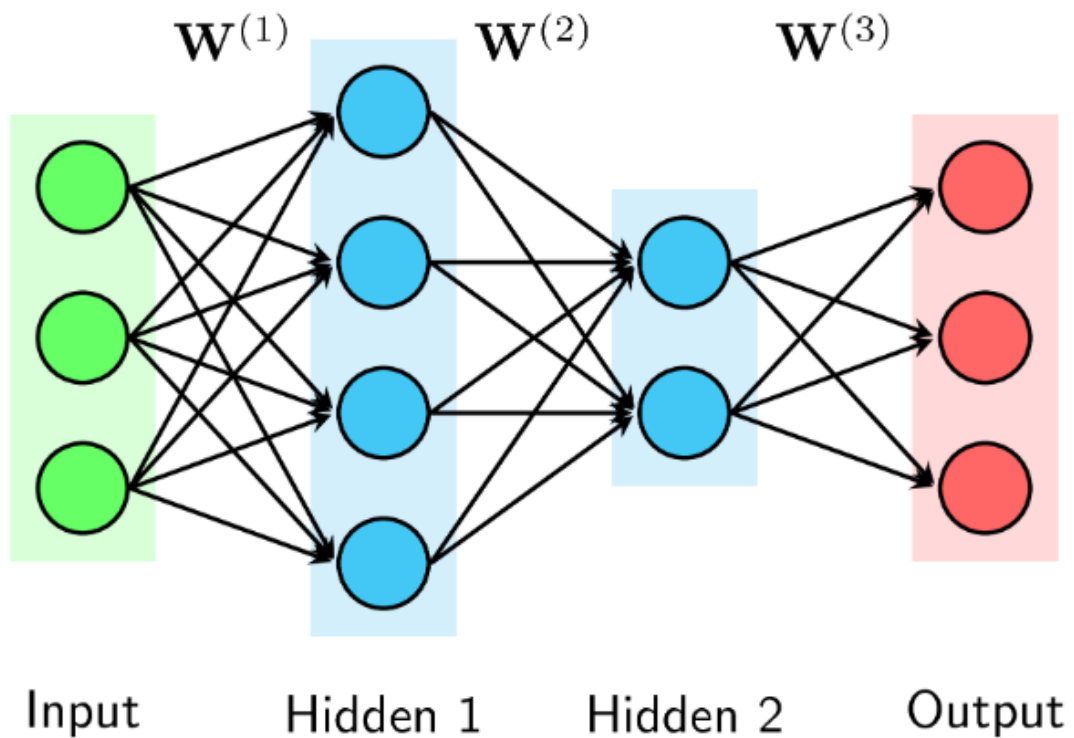
$$a_i^{(l)} = f(\mathbf{w}_i^{(l)T} \mathbf{a}^{(l-1)} + b_i^{(l)})$$

Trong đó $f(\cdot)$ là một (nonlinear) activation function. Ở dạng vector, biểu thức bên trên được viết là

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})$$

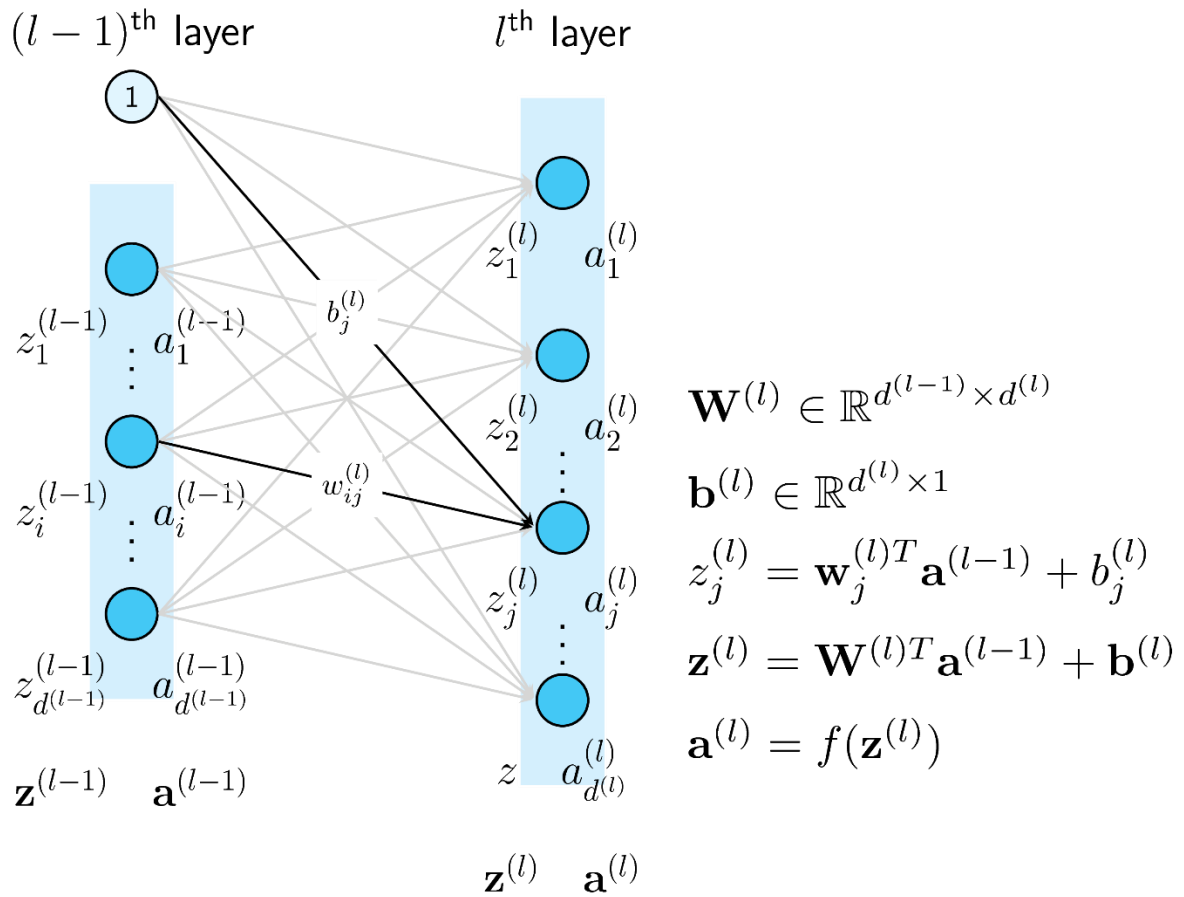
- Tạo ra kết quả cuối cùng
- Số lượng neuron phụ thuộc vào bài toán:

- Phân loại nhị phân: 1 neuron
- Phân loại nhiều lớp: số neuron = số lớp
- Hồi quy: số neuron = số biến cần dự đoán



Hình 12. MLP với hai hidden layers (các biases đã bị ẩn).

Số lượng layer trong một MLP được tính bằng số hidden layers cộng với 1. Tức là khi đếm số layers của một MLP, không tính input layers. Số lượng layer trong một MLP thường được ký hiệu là L . Trong Hình trên đây, $L=3$.



Hình 13. Các ký hiệu sử dụng trong MLP.

Một *node* hình tròn trong một layer được gọi là một unit. Unit ở các input layer, hidden layers, và output layer được lần lượt gọi là input unit, hidden unit, và output unit. Đầu vào của các hidden layer được ký hiệu bởi \mathbf{z} , đầu ra của mỗi unit thường được ký hiệu là \mathbf{a} (thể hiện *activation*, tức giá trị của mỗi unit sau khi ta dùng activation function lên \mathbf{z}). Đầu ra của unit thứ i trong layer thứ l được ký hiệu là $a_i^{(l)}$. Giả sử thêm rằng số unit trong layer thứ l (không tính bias) là $d^{(l)}$. Vector biểu diễn output của layer thứ l được ký hiệu là $\mathbf{a}^{(l)} \in \mathbb{R}^{d^{(l)}}$.

Có L ma trận trọng số cho một MLP có L layers. Các ma trận này được ký hiệu là $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$, $l = 1, 2, \dots, L$ trong đó $\mathbf{W}^{(l)}$ thể hiện các *kết nối* từ layer thứ $l-1$ tới layer thứ l (nếu ta coi input layer là layer thứ 0). Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của layer thứ $(l-1)$ tới node thứ j của layer thứ l . Các biases của layer thứ l được ký hiệu là $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$.

Tập hợp các weights và biases lần lượt được ký hiệu là \mathbf{W} và \mathbf{b} .

2.3.3. Cách thức thực thi

```
# Hỗ trợ cả Python 2 và Python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt

N = 100 # số điểm cho mỗi lớp
d0 = 2 # số chiều của dữ liệu
C = 3 # số lớp
X = np.zeros((d0, N*C)) # ma trận dữ liệu (mỗi cột là một điểm dữ liệu)
y = np.zeros(N*C, dtype='uint8') # nhãn của các lớp

# Thay xrange bằng range để tương thích với Python 3
for j in range(C):
    ix = range(N*j, N*(j+1))
    r = np.linspace(0.0, 1, N) # bán kính
    t = np.linspace(j*4, (j+1)*4, N) + np.random.randn(N)*0.2 # góc theta
    X[:,ix] = np.c_[r*np.sin(t), r*np.cos(t)].T
    y[ix] = j

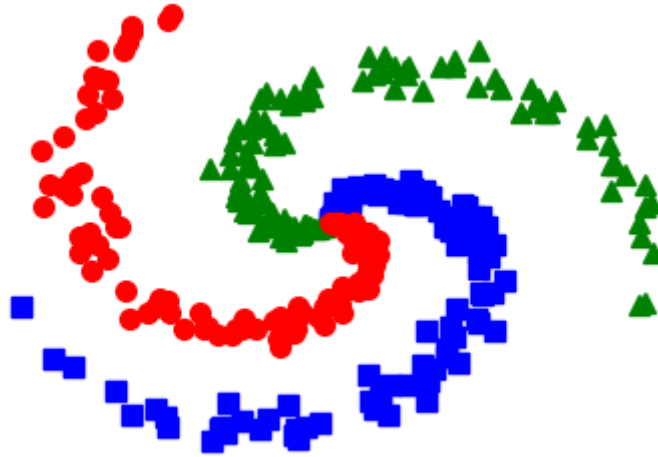
# Vẽ dữ liệu:
plt.scatter(X[:,N], X[:,1], c=y[:N], s=40, cmap=plt.cm.Spectral)

# Vẽ điểm dữ liệu cho từng lớp
plt.plot(X[0, :N], X[1, :N], 'bs', markersize = 7) # Lớp 1: hình vuông màu xanh
plt.plot(X[0, N:2*N], X[1, N:2*N], 'g^', markersize = 7) # Lớp 2: tam giác màu xanh lá
plt.plot(X[0, 2*N:], X[1, 2*N:], 'ro', markersize = 7) # Lớp 3: hình tròn màu đỏ

# Thiết lập giới hạn trục
plt.xlim([-1.5, 1.5])
plt.ylim([-1.5, 1.5])

# Ẩn các điểm đánh dấu trên trục
cur_axes = plt.gca() # lấy trục hiện tại
cur_axes.axes.get_xaxis().set_ticks([]) # ẩn điểm đánh dấu trục x
cur_axes.axes.get_yaxis().set_ticks([]) # ẩn điểm đánh dấu trục y

# Lưu và hiển thị hình
plt.savefig('EX.png', bbox_inches='tight', dpi = 600) # lưu hình với độ phân giải cao
plt.show() # hiển thị hình
```



Hình 14. Tập dữ liệu 3 lớp theo hình xoắn ốc

Tạo ra một tập dữ liệu gồm 3 lớp (classes) với các điểm dữ liệu được sắp xếp theo hình xoắn ốc (spiral). Mỗi lớp có 100 điểm, và mỗi điểm được biểu diễn trong không gian 2 chiều. Các điểm được tạo ra bằng cách sử dụng tọa độ cực (r, θ) và sau đó được chuyển đổi sang tọa độ Descartes (x, y) .

```

d0 = 2
d1 = h = 200 # size of hidden layer
d2 = C = 3

# Khởi tạo tham số
W1 = 0.01*np.random.randn(d0, d1)
b1 = np.zeros((d1, 1))
W2 = 0.01*np.random.randn(d1, d2)
b2 = np.zeros((d2, 1))

# Chuyển đổi labels sang one-hot encoding
Y = convert_labels(y, C)
N = X.shape[1]
eta = 1 # learning rate

# Training loop
for i in range(10000): # Thay xrange bằng range cho Python 3
    # Feedforward
    Z1 = np.dot(W1.T, X) + b1
    A1 = np.maximum(Z1, 0) # ReLU activation
    Z2 = np.dot(W2.T, A1) + b2
    Yhat = softmax(Z2)

    # Tính loss
    loss = cost(Y, Yhat)

    if i % 1000 == 0:
        print("iter %d, loss: %f" %(i, loss))

    # Backpropagation
    E2 = (Yhat - Y)/N
    dW2 = np.dot(A1, E2.T)
    db2 = np.sum(E2, axis=1, keepdims=True)
    E1 = np.dot(W2, E2)
    E1[Z1 <= 0] = 0 # ReLU gradient
    dW1 = np.dot(X, E1.T)
    db1 = np.sum(E1, axis=1, keepdims=True)

    # Gradient Descent update
    W1 += -eta*dW1
    b1 += -eta*db1
    W2 += -eta*dW2
    b2 += -eta*db2

```

```

iter 0, loss: 1.098613
iter 1000, loss: 0.113153
iter 2000, loss: 0.052247
iter 3000, loss: 0.036119
iter 4000, loss: 0.029689
iter 5000, loss: 0.026148
iter 6000, loss: 0.023806
iter 7000, loss: 0.022121
iter 8000, loss: 0.020792
iter 9000, loss: 0.019734

```

Loss giảm nhanh trong 1000 iterations đầu tiên (từ 1.09 xuống 0.15)

Sau đó tiếp tục giảm nhưng chậm hơn

Đến iteration thứ 9000, loss đã giảm xuống còn khoảng 0.02

Loss function hội tụ tốt, cho thấy model đang học hiệu quả

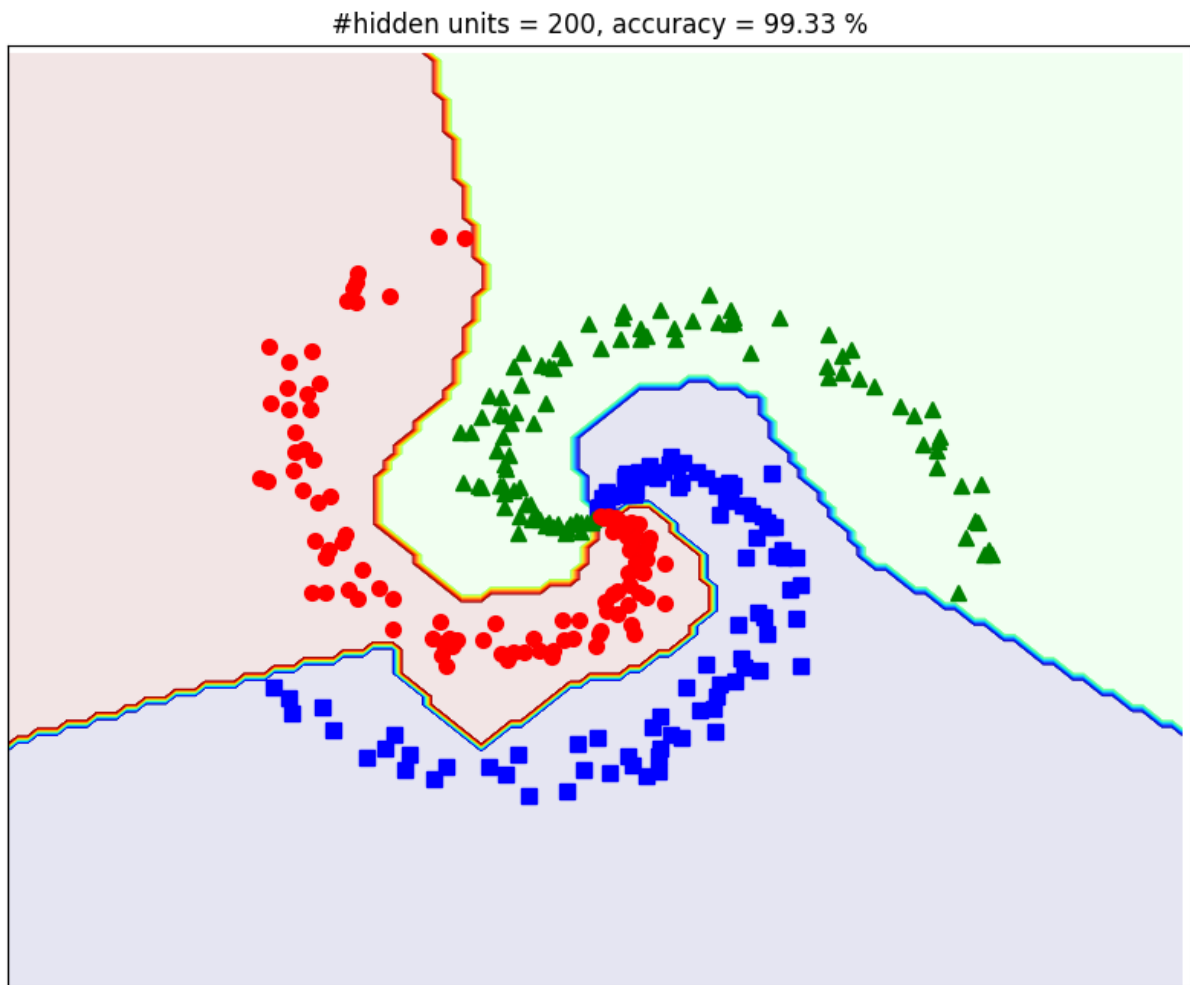
```
# Predict và tính accuracy
Z1 = np.dot(W1.T, X) + b1
A1 = np.maximum(Z1, 0)
Z2 = np.dot(W2.T, A1) + b2
predicted_class = np.argmax(Z2, axis=0)
acc = 100*np.mean(predicted_class == y)
print('training accuracy: %.2f %%' % acc)
```

```
training accuracy: 99.33 %
```

Model phân loại đúng 298/ 300 điểm dữ liệu

Chỉ có 2 điểm bị phân loại sai

Độ chính xác rất cao cho bài toán phân loại phi tuyến này



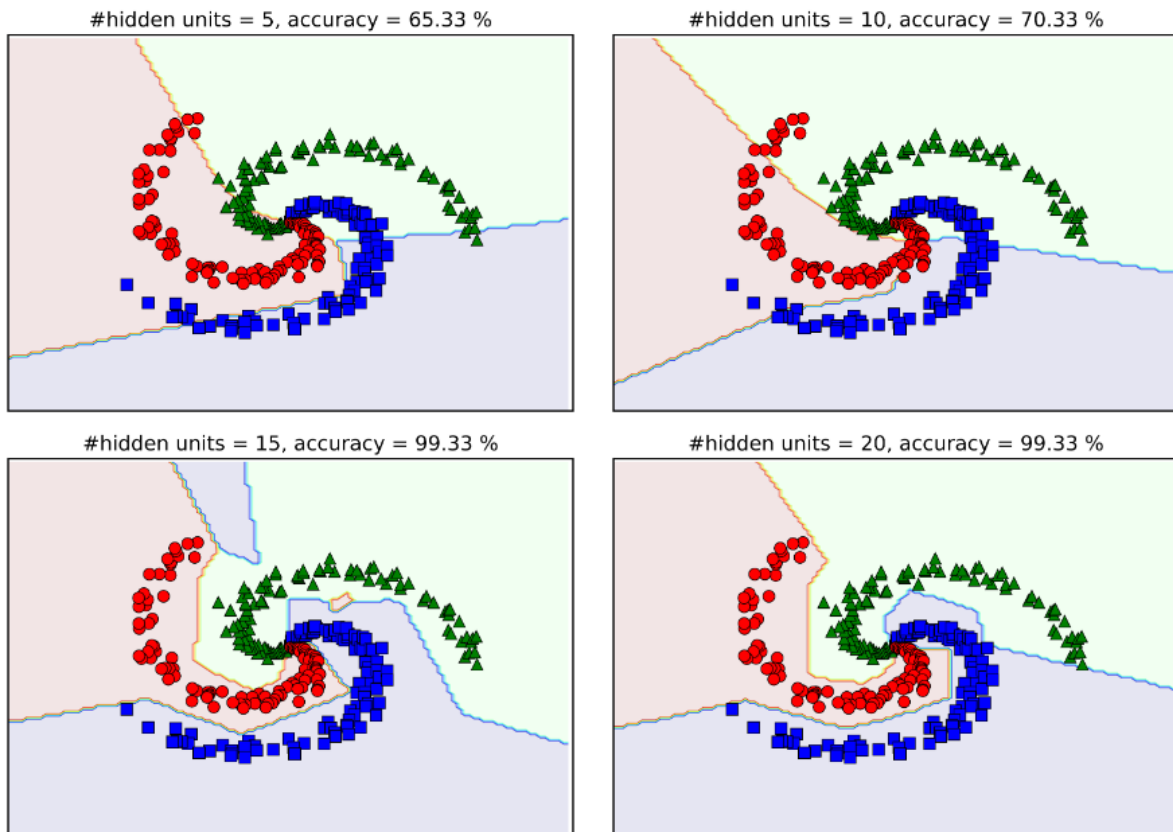
Hình 15. Kết quả hiển thị 3 vùng màu khác nhau cho 3 classes

Các boundary giữa các class có dạng cong (nonlinear)

Các điểm dữ liệu:

- Hình vuông màu xanh dương: Class 1
- Hình tam giác màu xanh lá: Class 2
- Hình tròn màu đỏ: Class 3

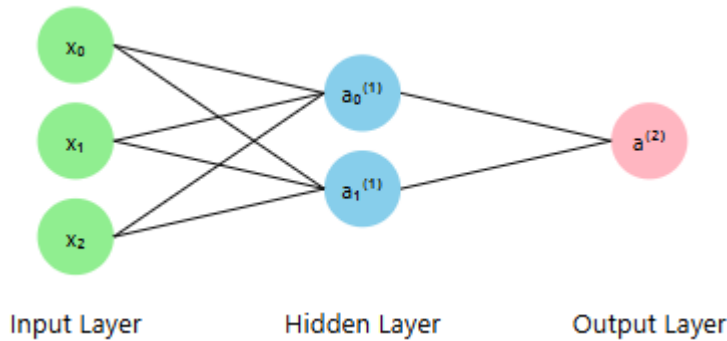
Decision boundaries mượt mà và phù hợp với phân bố dữ liệu



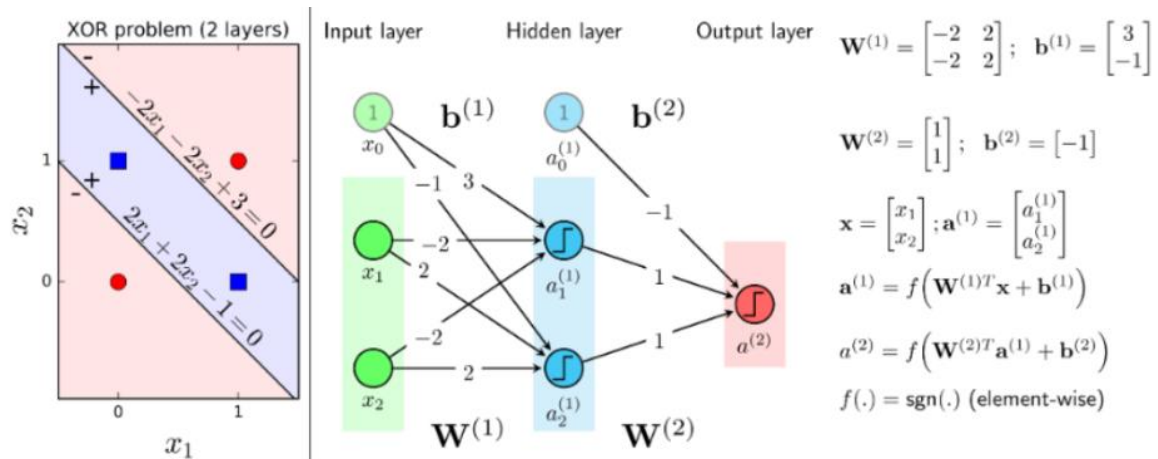
Hình 16. Kết quả với số lượng units trong hidden layer là khác nhau.

- Khi số lượng hidden units tăng lên, độ chính xác của mô hình tạo được cũng tăng lên.
- Với $d1 = 5$, đường phân định giữa ba classes gần như là đường thẳng.
- Với $d1 = 15$, mặc dù kết quả đã đạt 99.33%, vẫn có một vùng đỏ nhỏ nằm giữa nhánh màu lục và màu lam, và một vùng màu lam khá lớn giữa màu đỏ và lục. Khi một điểm dữ liệu test rơi vào những vùng này, nó sẽ bị phân loại sai.
- Với $d1 = 20$, kết quả nhận được đã tương đối giống với $d1 = 100$. Mặc dù các đường boundary không được trơn tru cho lắm.

2.3.4. Ví dụ giải tay



Mạng MLP được sử dụng để giải quyết bài toán XOR, một bài toán kinh điển trong học máy

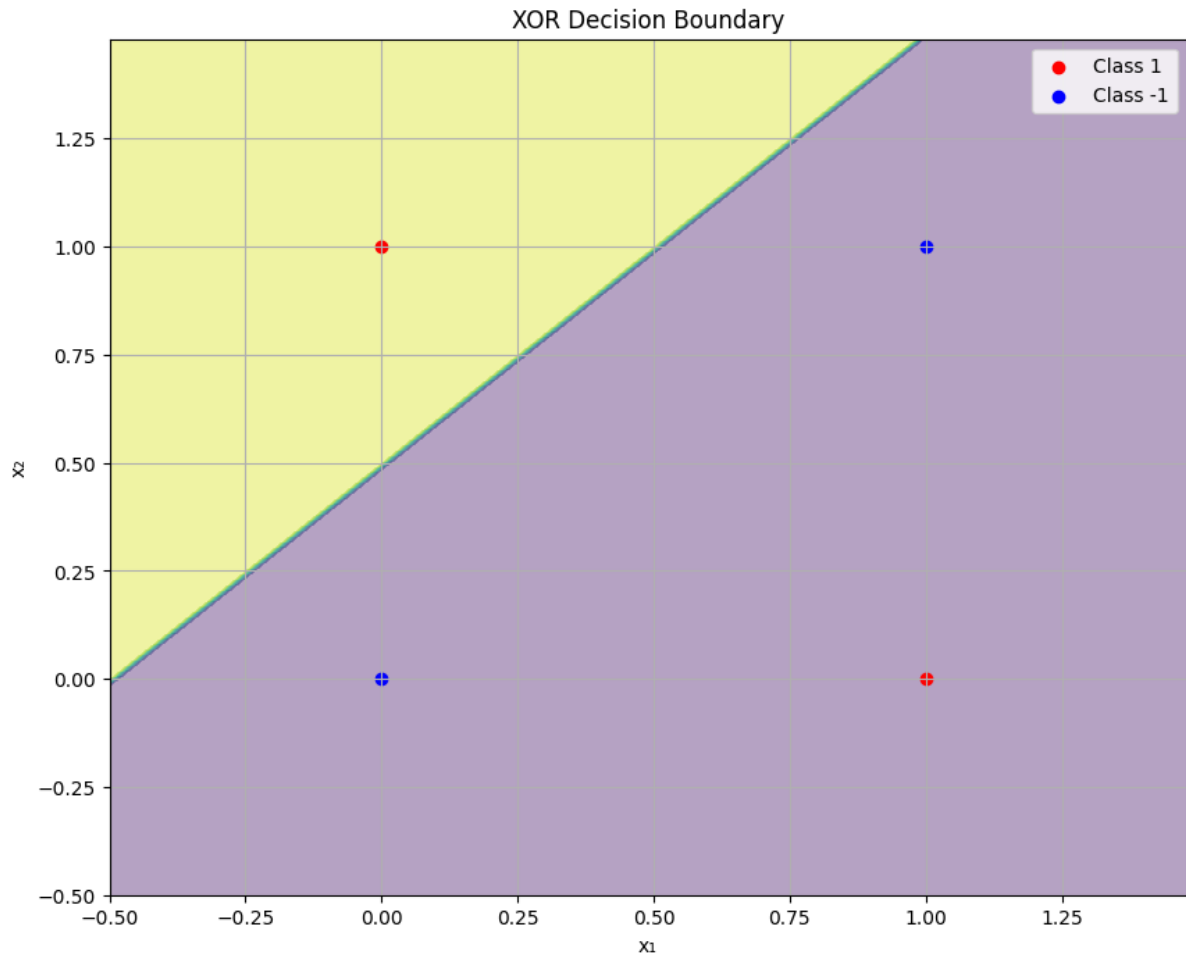


Hình 17. Mạng MLP giải quyết bài toán XOR

$z_0^{(1)} = (-2 \times 1) + (2 \times 0) + 3$ [nhân $\mathbf{W}^{(1)}$ hàng 1 với \mathbf{x} và cộng $\mathbf{b}^{(1)}[0]$] = $-2 + 0 + 3 = 1$ $a_0^{(1)} = \text{sgn}(1) = 1$ [áp dụng hàm kích hoạt]
 $z_1^{(1)} = (-2 \times 1) + (2 \times 0) + (-1)$ [nhân $\mathbf{W}^{(1)}$ hàng 2 với \mathbf{x} và cộng $\mathbf{b}^{(1)}[1]$] = $-2 + 0 + (-1) = -3$ $a_1^{(1)} = \text{sgn}(-3) = -1$ [áp dụng hàm kích hoạt]
 $z^{(2)} = (1 \times 1) + (1 \times (-1)) + (-1)$ [nhân $\mathbf{W}^{(2)}$ với $[a_0^{(1)}, a_1^{(1)}]$ và cộng $\mathbf{b}^{(2)}$] = $1 + (-1) + (-1) = -1$ $a^{(2)} = \text{sgn}(-1) = -1$ [áp dụng hàm kích hoạt]

- XOR(1,0) = 1 trong logic Boolean
- Trong mạng này ta biểu diễn:
 - Giá trị 1 (Boolean) = 1 (Neural Network)
 - Giá trị 0 (Boolean) = -1 (Neural Network)

Tương tự, kiểm tra các trường hợp còn lại:



Hình 18. Kết quả bài toán XOR

3. Phương pháp Histogram of Oriented Gradients (HOG)

3.1. Khái niệm

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, là phương pháp trích xuất đặc trưng trong xử lý ảnh, dựa trên việc phân tích sự thay đổi cường độ sáng tại các pixel theo hướng và độ lớn.

3.2. Nguyên lý hoạt động

Nguyên lý hoạt động của thuật toán là dựa trên biểu diễn vector histogram của độ lớn gradient theo các bins của phương gradient áp dụng trên những vùng ảnh cục bộ. Các phương pháp chuẩn hóa được áp dụng giúp vector histogram tổng hợp trở nên bất biến với sự thay đổi về cường độ màu sắc của các bức ảnh có cùng nội dung nhưng khác nhau về cường độ màu sắc.

Đầu tiên hình ảnh được chia thành 1 lưới ô vuông và trên đó xác định rất nhiều các vùng cục bộ liên kề hoặc chồng lấn lên nhau. Các vùng này tương tự như những vùng hình ảnh cục bộ tính tích chập trong thuật toán CNN. Một vùng cục bộ bao gồm nhiều ô cục bộ (trong thuật toán HOG là 4) có kích thước là 8x8 pixels. Sau đó, một biểu đồ histogram thống kê độ lớn gradient được tính toán trên mỗi ô cục bộ. Bộ mô tả HOG (HOG descriptor) được tạo thành bằng cách nối liền (concatenate) 4 véc tơ histogram ứng với mỗi ô thành một véc tơ tổng hợp. Để cải thiện độ chính xác, mỗi giá trị của véc tơ histogram trên vùng cục bộ sẽ được chuẩn hóa theo norm chuẩn bậc 2 hoặc bậc 1. Phép chuẩn hóa này nhằm tạo ra sự bất biến tốt hơn đối với những thay đổi trong chiếu sáng và đổ bóng.

Bộ mô tả HOG có một vài lợi thế chính so với các bộ mô tả khác. Vì nó hoạt động trên các ô cục bộ, nó bất biến đối với các phép biến đổi hình học, thay đổi độ sáng. Hơn nữa, như Dalal và Triggs đã phát hiện ra, khi sử dụng phép chuẩn hóa trên vùng cục bộ sẽ cho phép chuyển động cơ thể của người đi bộ được loại bỏ miễn là họ duy trì được tư thế đứng thẳng. Do đó, bộ mô tả HOG đặc biệt phù hợp để phát hiện con người trong hình ảnh. Trong một số bài toán dữ liệu nhỏ, HOG có thể hiệu quả hơn CNN, bởi nó ít tốn tài nguyên và tránh overfitting.

Qua đó thấy được HOG mặc dù là phương pháp cũ nhưng vẫn rất hiệu quả trong nhiều bài toán. Tùy từng tình huống mà có thể sử dụng thuật toán HOG chứ không nhất thiết phải áp dụng một mô hình deep learning với hàng triệu tham số thì mới mang lại độ chính xác cao.

3.3. Các bước thực thi quy trình trích xuất đặc trưng HOG

Bước 1: Tiền xử lý

- Chuyển ảnh gốc sang ảnh grayscale (nếu ảnh gốc là ảnh màu RGB)
- Cân chỉnh kích thước tất cả các hình ảnh về cùng kích thước.

```
# Đảm bảo ảnh là grayscale
if len(image.shape) > 2:
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Bước 2: Tính toán hình ảnh Gradient

- Sử dụng bộ lọc Sobel để tính toán gradient của từng pixel theo trục x và y
- Gradient GGG được xác định:

$$G = \sqrt{(G_x^2 + G_y^2)}, \theta = \arctan \frac{G_y}{G_x}$$

| Trong đó:

- G: Độ lớn gradient.
- θ : Hướng gradient.

```
# Tính gradient theo x và y
gx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
gy = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# Tính magnitude và orientation của gradient
magnitude = np.sqrt(gx**2 + gy**2) # Đây là G (độ lớn gradient)
orientation = np.arctan2(gy, gx) * 180 / np.pi # Đây là  $\theta$  (hướng gradient)
```

Bước 3: Tính vector đặc trưng cho từng ô (cells)

- Chia hình ảnh thành các ô (cells), mỗi ô có kích thước 8×8 pixels.
- Với mỗi ô, tính histogram của hướng gradient, chia thành NNN hướng (bins), ví dụ $N=9$ hướng từ 0° đến 180°
- Kết quả: Vector đặc trưng cho mỗi ô.

```

# Tính số cells theo mỗi chiều
cells_y = height // cell_size
cells_x = width // cell_size

# Khởi tạo histogram
histogram = np.zeros((cells_y, cells_x, num_bins))

# Góc của các bins
bin_size = 180 // num_bins

# Lặp qua từng cell
for i in range(cells_y):
    for j in range(cells_x):
        # Lấy các giá trị trong cell hiện tại
        cell_magnitudes = magnitude[i*cell_size:(i+1)*cell_size,
                                     j*cell_size:(j+1)*cell_size]
        cell_orientations = orientation[i*cell_size:(i+1)*cell_size,
                                        j*cell_size:(j+1)*cell_size]

        # Tính histogram cho cell
        for y in range(cell_size):
            for x in range(cell_size):
                orient = cell_orientations[y, x]
                mag = cell_magnitudes[y, x]

```

```

# Đưa góc về khoảng 0-180
if orient < 0:
    orient += 180

# Tính bin index
bin_index = int(orient // bin_size)
if bin_index >= num_bins:
    bin_index = num_bins - 1

histogram[i, j, bin_index] += mag

```

Chuẩn hóa khối (blocks)

- Gom nhóm 2×2 ô thành một khối (block)
- Chuẩn hóa vector đặc trưng của các ô trong khối để giảm tác động của điều kiện ánh sáng

```

cells_y, cells_x, bins = histogram.shape
blocks_y = cells_y - block_size + 1
blocks_x = cells_x - block_size + 1

normalized_blocks = []

# Lặp qua các block
for i in range(blocks_y):
    for j in range(blocks_x):
        # Lấy block hiện tại
        block = histogram[i:i+block_size, j:j+block_size, :]

        # Làm phẳng block thành vector
        block_vector = block.ravel()

        # Chuẩn hóa L2
        norm = np.sqrt(np.sum(block_vector**2) + 1e-6)
        normalized_blocks.append(block_vector / norm)

```

Tính toán vector HOG

- Ghép các vector đặc trưng từ tất cả các khối lại thành một vector HOG duy nhất cho hình ảnh
 - o Ghép các histogram đã chuẩn hóa
 - o Tạo vector đặc trưng cuối cùng mô tả hình dạng đối tượng

```

def compute_hog_features(image, cell_size=8, block_size=2, num_bins=9):
    """Tính toán HOG features cho ảnh."""
    # Đảm bảo ảnh là grayscale
    if len(image.shape) > 2:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Tính gradient
    gx, gy, magnitude, orientation = compute_gradient(image)

    # Tạo histogram
    histogram = create_histogram(magnitude, orientation, cell_size, num_bins)

    # Chuẩn hóa blocks
    hog_features = normalize_blocks(histogram, block_size)

    return hog_features

```

Ví dụ:

Tạo một ảnh grayscale đơn giản kích thước 4x4 pixels. Mỗi pixel có giá trị cường độ sáng khác nhau (từ 10 đến 80)

```
# Tạo ảnh mẫu 4x4
sample_image = np.array([
    [10, 30, 40, 80],
    [20, 25, 30, 50],
    [30, 45, 50, 55],
    [15, 20, 35, 40]
], dtype=np.uint8)
```

Thay đổi kích thước ảnh: Resize ảnh từ 4x4 lên 16x16 pixels. Việc này cần thiết vì để tính HOG features, ảnh cần có kích thước đủ lớn để chia thành các cells (thường là 8x8 pixels)

```
resized_image = cv2.resize(sample_image, (16, 16), interpolation=cv2.INTER_CUBIC)
```

Tính toán HOG features:

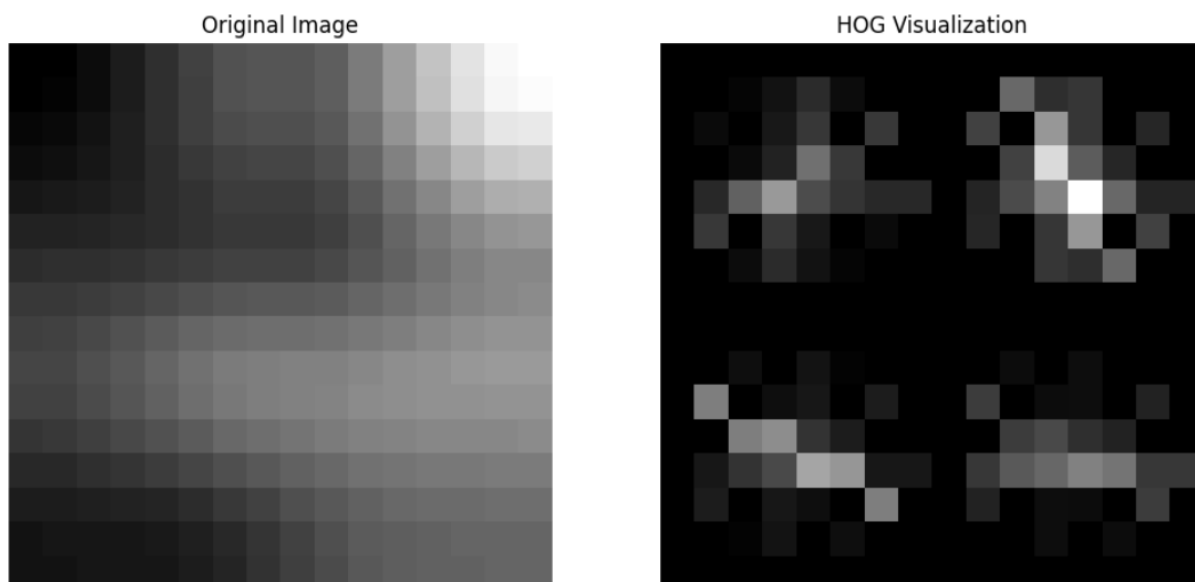
- Áp dụng thuật toán HOG đã định nghĩa lên ảnh đã resize

```
hog_features = compute_hog_features(resized_image)
print("HOG features shape:", hog_features.shape)
print("HOG features:", hog_features)
```

- In ra:
 - Shape của vector HOG features (cho biết độ dài của vector đặc trưng)
 - Giá trị của các features

Hiển thị kết quả:

- Vẽ và hiển thị hai hình:
 - Ảnh gốc (đã resize)
 - Visualization của HOG features để có thể nhìn thấy cách HOG biểu diễn các gradient trong ảnh



Hình 19. Kết quả quy trình trích xuất đặc trưng của HOG

```
HOG features shape: (1, 36)
HOG features: [[0.20235781 0.05475783 0.05953241 0.24852991 0.15494776 0.01418005
0.02058018 0.02818744 0.09577286 0.25295005 0.02619634 0.08430834
0.06935371 0.17457912 0.10883705 0.20080912 0.46596861 0.2423043
0.09272874 0.01383946 0.06415278 0.06705748 0.09146573 0.48938181
0.10546207 0.0848227 0.0546602 0.00583968 0.0325989
0.13801073 0.22615414 0.22947087 0.08256156 0.06113169 0. ]]
```

HOG features shape: (1, 36)

- Vector đặc trưng HOG có kích thước 36 chiều
- Con số 36 này đến từ việc tính toán:
 - Ảnh 16x16 được chia thành các cells 8x8 → có 4 cells (2x2)
 - Mỗi block 2x2 cells → có 1 block
 - Mỗi cell có histogram 9 bins
 - Do đó: $1 \text{ block} \times (2 \times 2 \text{ cells}) \times 9 \text{ bins} = 36 \text{ features}$

Original Image (ảnh bên trái)

- Hiển thị ảnh mẫu đã được resize lên 16x16 pixels
- Các giá trị pixel được thể hiện qua độ sáng tối:
 - Vùng sáng (màu xám nhạt) thể hiện giá trị pixel cao
 - Vùng tối (màu xám đậm) thể hiện giá trị pixel thấp

HOG Visualization (ảnh bên phải)

- Thể hiện trực quan các gradient đã được tính toán

- Các vùng sáng thể hiện:
 - Nơi có sự thay đổi mạnh về cường độ sáng (gradient lớn)
 - Hướng của gradient (sự thay đổi) trong ảnh
- Pattern trong ảnh HOG cho thấy:
 - Cạnh và góc được phát hiện
 - Sự thay đổi về cường độ sáng theo các hướng khác nhau

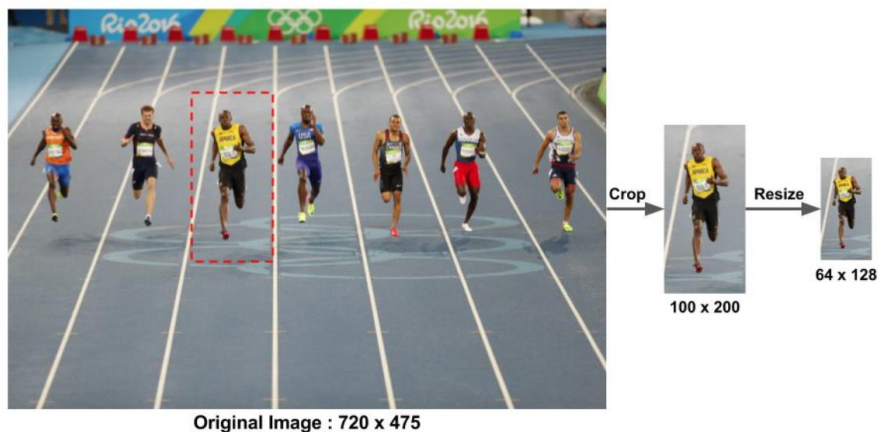
HOG features (dãy số)

- Các giá trị số thể hiện magnitude của gradient theo các hướng khác nhau
- Đã được chuẩn hóa (normalize) nên các giá trị nằm trong khoảng $[0,1]$
- Mỗi số thể hiện "mức độ" thay đổi cường độ sáng theo một hướng cụ thể trong block

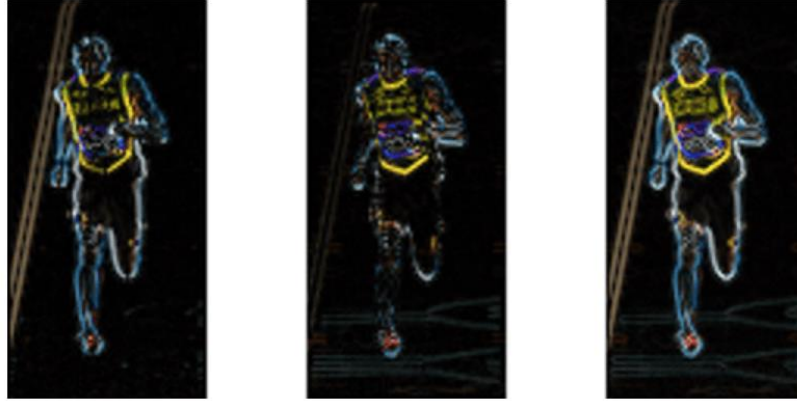
3.4. Ví dụ giải tay

Một hình ảnh lớn có kích thước 720×475 . Nhóm đã chọn một bản vá có kích thước 100×200 để tính toán mô tả tính năng HOG. Bản vá này được cắt ra khỏi hình ảnh và thay đổi kích thước thành 64×128 . Bây giờ chúng ta đã sẵn sàng để tính toán mô tả HOG cho bản vá hình ảnh này

Bước 1: Tiền xử lý hình ảnh:



Bước 2: Tính toán Gradient

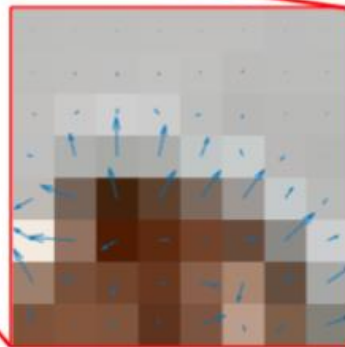
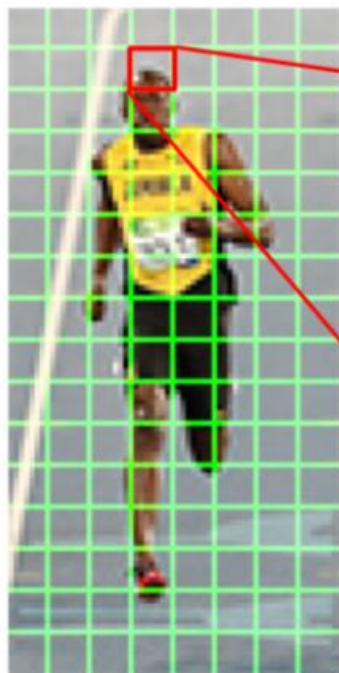
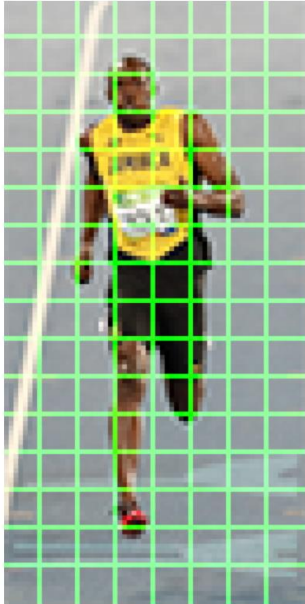


Trái : Giá trị tuyệt đối của x-gradient Trung tâm: Giá trị tuyệt đối của gradient y Phải: Độ lớn của gradient

Hình ảnh gradient đã loại bỏ rất nhiều thông tin không cần thiết (ví dụ: nền màu không đổi), nhưng được đánh dấu các đường viền. Nói cách khác, bạn có thể nhìn vào hình ảnh gradient mà vẫn dễ dàng nói rằng có một người trong ảnh. Ở mỗi pixel, gradient có độ lớn và hướng. Đối với hình ảnh màu, độ dốc của ba kênh được đánh giá (như trong hình trên). Độ lớn của gradient tại một pixel là độ lớn của độ dốc của ba kênh và góc là góc tương ứng với gradient tối đa

Bước 3: Tính biểu đồ của các gradient trong 8×8 ô

Bản vá hình ảnh 8×8 chứa giá trị $8 \times 8 \times 3 = 192$ pixel. Độ dốc của bản vá này chứa 2 giá trị (độ lớn và hướng) trên mỗi pixel cộng lại là $8 \times 8 \times 2 = 128$ số. HOG ban đầu được sử dụng để phát hiện người đi bộ. 8×8 ô trong ảnh người đi bộ được tỷ lệ thành 64×128 đủ lớn để chụp các đặc điểm thú vị (ví dụ: khuôn mặt, đỉnh đầu, v.v.). Biểu đồ về cơ bản là một vectơ (hoặc một mảng) gồm 9 thùng (số) tương ứng với các góc 0, 20, 40, 60 ... 160



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

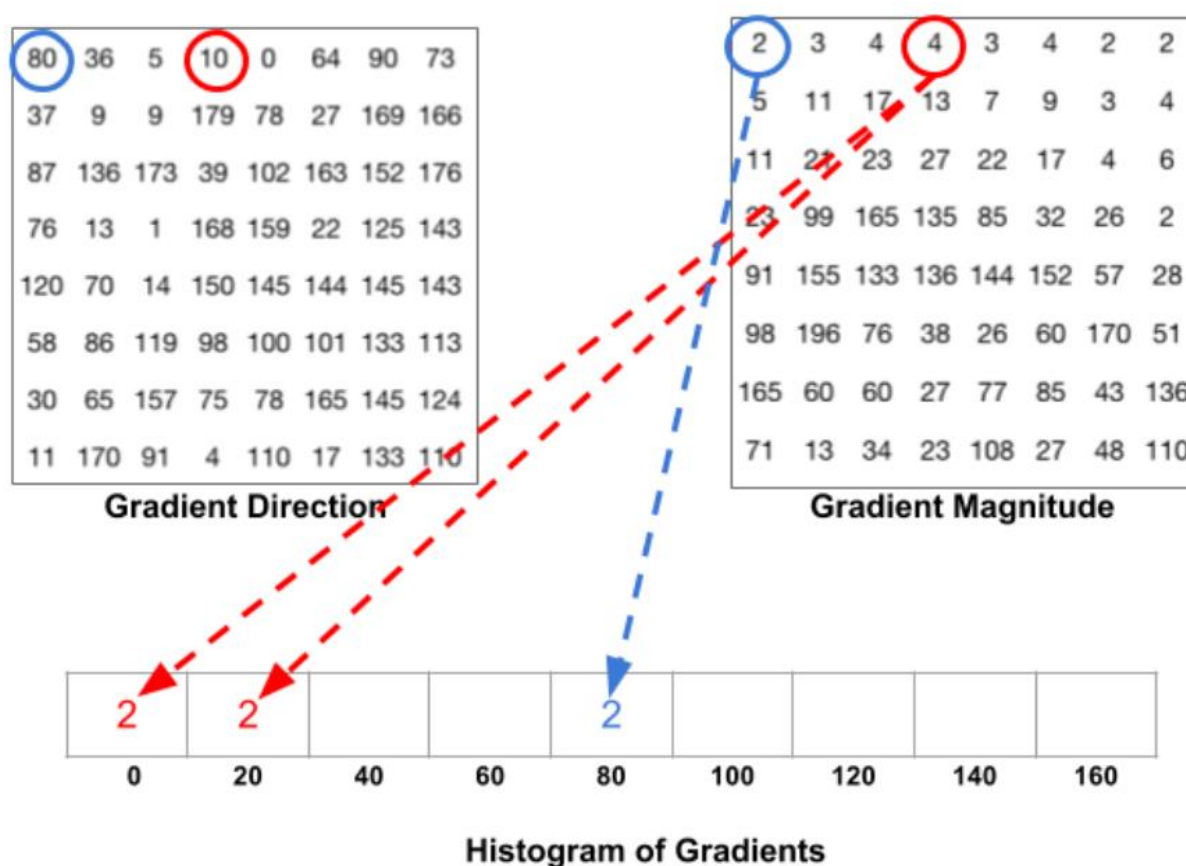
Hình ảnh vận động viên được chia thành các lưới ô vuông, mỗi ô vuông có kích thước 8x8 pixels. Trên mỗi ô thực hiện tính đạo hàm thông qua bộ lọc Sobel để thu được 2 ma trận bên phải là gradient magnitude và gradient direction.

Hình ảnh được chia thành một lưới ô vuông mà mỗi một ô có kích thước 8x8 pixels. Như vậy có tổng cộng 64 ô pixels tương ứng với mỗi ô. Trên mỗi một ô trong 64 pixels sẽ cần tính ra 2 tham số đó là độ lớn gradient (gradient magnitude) và phương gradient (gradient direction). Tổng cộng $8 \times 8 \times 2 = 128$ giá trị cần tính bao gồm 64 giá trị gradient magnitude và 64 giá trị gradient direction như ma trận

Véc tơ histogram sẽ được tạo ra như sau:

Sắp xếp các giá trị phương gradient theo thứ tự từ nhỏ đến lớn và chia chúng vào 9 bins. Độ lớn của phương gradient sẽ nằm trong khoảng $[0, 180]$ nên mỗi bins sẽ có độ dài là 20. Mỗi một phương gradient sẽ ghép cặp với một độ lớn gradient ở cùng vị trí tọa độ. Khi biết được phương gradient thuộc bins nào trong véc tơ bins, điền vào giá trị giá trị của độ lớn gradient vào chính bin đó

Chẳng hạn trong hình bên dưới ô được khoanh trong hình tròn viền xanh tương ứng với phương gradient là 80 và độ lớn gradient là 2. Khi đó tại véc tơ bins của HOG, phương gradient bằng 80 sẽ rơi vào vị trí thứ 5 nên tại ô này chúng ta điền giá trị 2 ứng với độ lớn gradient.



Hình 20. Mapping độ lớn gradients với các bins

Đầu mút là các giá trị chia hết cho độ rộng của một bin (chẳng hạn 0, 20, 40, ... là những đầu mút bin). Trong trường hợp độ lớn phương gradients không rơi vào các đầu mút, ta sẽ sử dụng linear interpolation để phân chia độ lớn gradient về 2 bins liền kề mà giá trị phương gradient rơi vào.

Ví dụ: giá trị phương gradient bằng x , cặp với độ lớn gradient bằng y . $x \in [x_0, x_1]$

tức là phương gradients rơi vào khoảng giữa bin $(l-1)$ và bin l . Khi đó tại 2 bins

$(l-1)$ l

và được điền vào giá trị cường độ theo công thức interpolation:

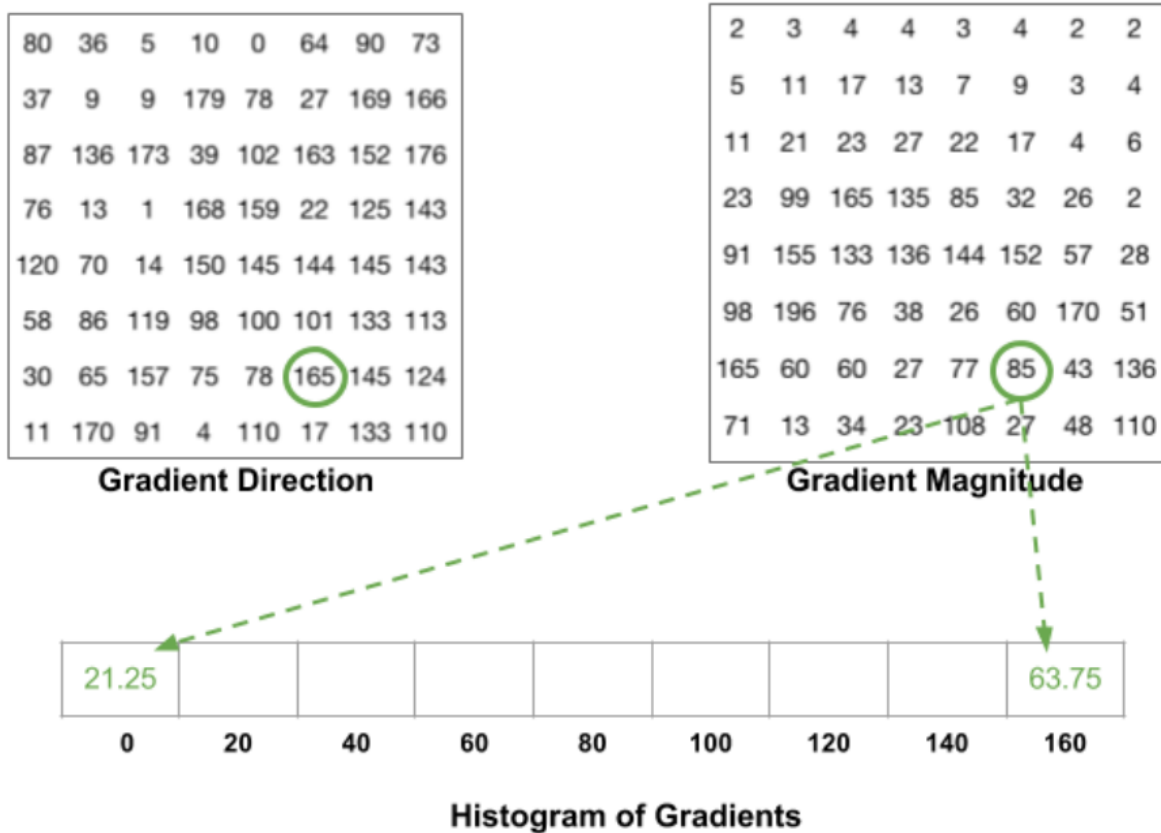
Giá trị tại bins $l - 1$

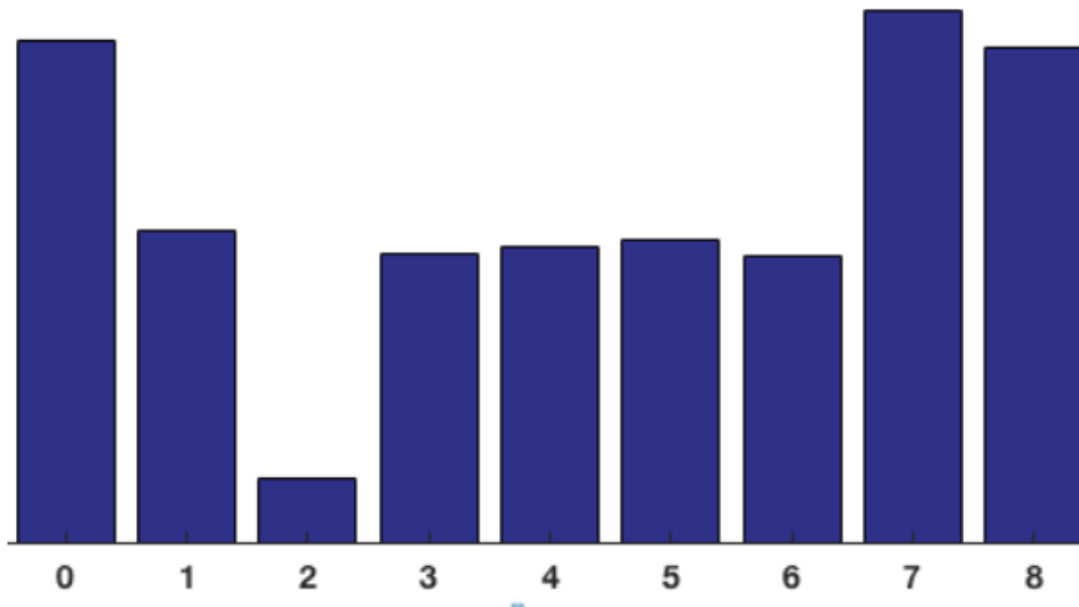
$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y$$

Giá trị tại bins l

$$x_l = \frac{(x - x_0)}{x_1 - x_0} * y$$

Với điểm được khoanh tròn bởi hình tròn màu xanh có phương gradient bằng 165 và độ lớn gradient bằng 85. Ta phân chia giá trị về các bins 0 (hoặc 180) và 160 các giá trị theo công thức interpolation bên trên. Kết quả cuối cùng chúng ta thu được là:



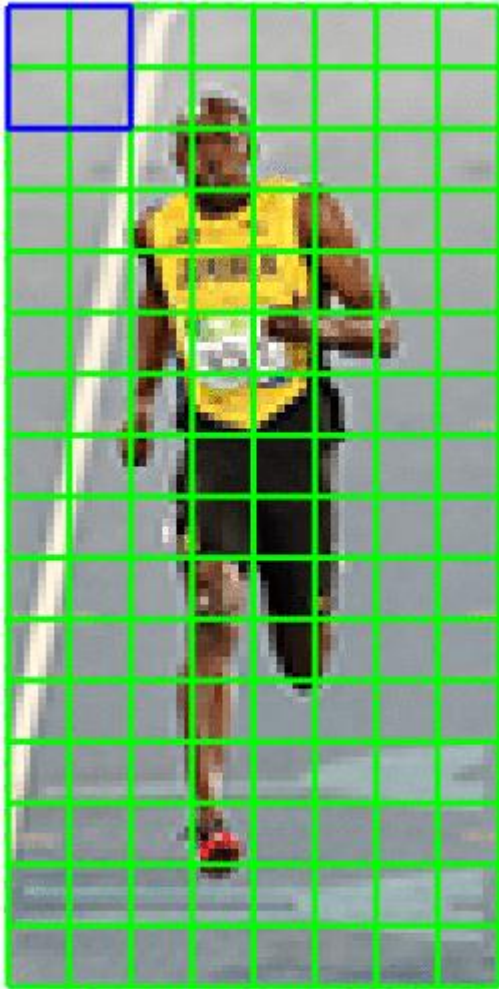


Hình 21. Biểu đồ Histogram of Gradient gồm 9 bins tương ứng với một ô vuông trong lưới ô vuông.

Chuẩn hóa véc tơ histogram theo block 16x16 chuẩn hóa norm chuẩn bậc 2

$$\text{normalize}(\mathbf{h}) = \frac{\mathbf{h}}{\|\mathbf{h}\|_2}$$

Quá trình chuẩn hóa sẽ thực hiện trên một block kích thước **2x2** trên lưới ô vuông ban đầu (mỗi ô kích thước **8x8** pixel). Như vậy chúng ta sẽ có 4 véc tơ histogram kích thước **1x9**, concatenate các véc tơ sẽ thu được véc tơ histogram tổng hợp kích thước là **1x36** và sau đó chuẩn hóa theo norm chuẩn bậc 2 trên véc tơ này. Việc di chuyển các window thực hiện tương tự như phép tích chập 2 chiều trong mạng CNN với $\text{step_size} = 8$ pixels. Phân chia thành lưới các ô vuông con, mỗi ô kích thước 8x8 pixel. Thực hiện chuẩn hóa véc tơ histogram trên các block gồm 2x2 (đơn vị ô) ứng với kích thước 16x16 pixel.



Tính toán HOG feature véc tơ sau khi chuẩn hóa các véc tơ histogram, concatenate các véc tơ 1×36 này thành một véc tơ lớn. Đây chính là véc tơ HOG đại diện cho toàn bộ hình ảnh. Chia thành lưới ô vuông kích thước 16×8 (mỗi ô 8×8). Quá trình tính toán HOG sẽ di chuyển 7 lượt theo chiều rộng và 15 lượt theo chiều cao. Như vậy sẽ có tổng cộng $7 \times 15 = 105$ patches, mỗi patch tương ứng với 1 véc tơ histograms 36 chiều. Do đó cuối cùng véc tơ HOG sẽ có kích thước là $105 \times 36 = 3780$ chiều. Đây là một véc tơ kích thước tương đối lớn nên có thể mô phỏng được đặc trưng của ảnh khá tốt. Đối với mỗi một ô trên lưới ô vuông, chúng ta biểu diễn phân phối HOG bao gồm nhóm 9 véc tơ chung gốc chiều dài bằng độ lớn gradient và góc bằng phương gradient. Khi đó chiều của nhóm các véc tơ sẽ tương đối giống với dáng của vận động viên trong ảnh, đặc biệt là tại các vị trí chân và tay. Biểu diễn nhóm véc tơ histogram trên các lưới ô vuông của hình ảnh gốc. Các phương véc tơ phổ biến là chiều dọc trùng với chiều bức ảnh.



Chương 3. Xây dựng bộ dữ liệu

👉 Nội dung trình bày quy trình thu thập, xử lý, và phân chia dữ liệu để tạo ra một bộ dữ liệu chất lượng, phục vụ cho việc huấn luyện và đánh giá mô hình phân loại chữ viết tay tiếng Việt.

- ❖ Thông tin về bộ dữ liệu gốc (VietData) bộ dữ liệu được sử dụng để thực nghiệm và làm tập huấn luyện.
- ❖ Quy trình thu thập dữ liệu bổ sung cho tập Test set
- ❖ Quy trình tiền xử lý dữ liệu

❖ Phân chia dữ liệu

1. Cách thu thập dữ liệu (Preprocess_data_to_test)

Công trình được hoàn thành tại: Viện Công nghệ Thông tin – Viện Khoa học và Công nghệ Việt Nam.

Tác giả: PHẠM ANH PHƯƠNG.

Ngày công bố: Hà Nội, ngày 04 tháng 6 năm 2010.

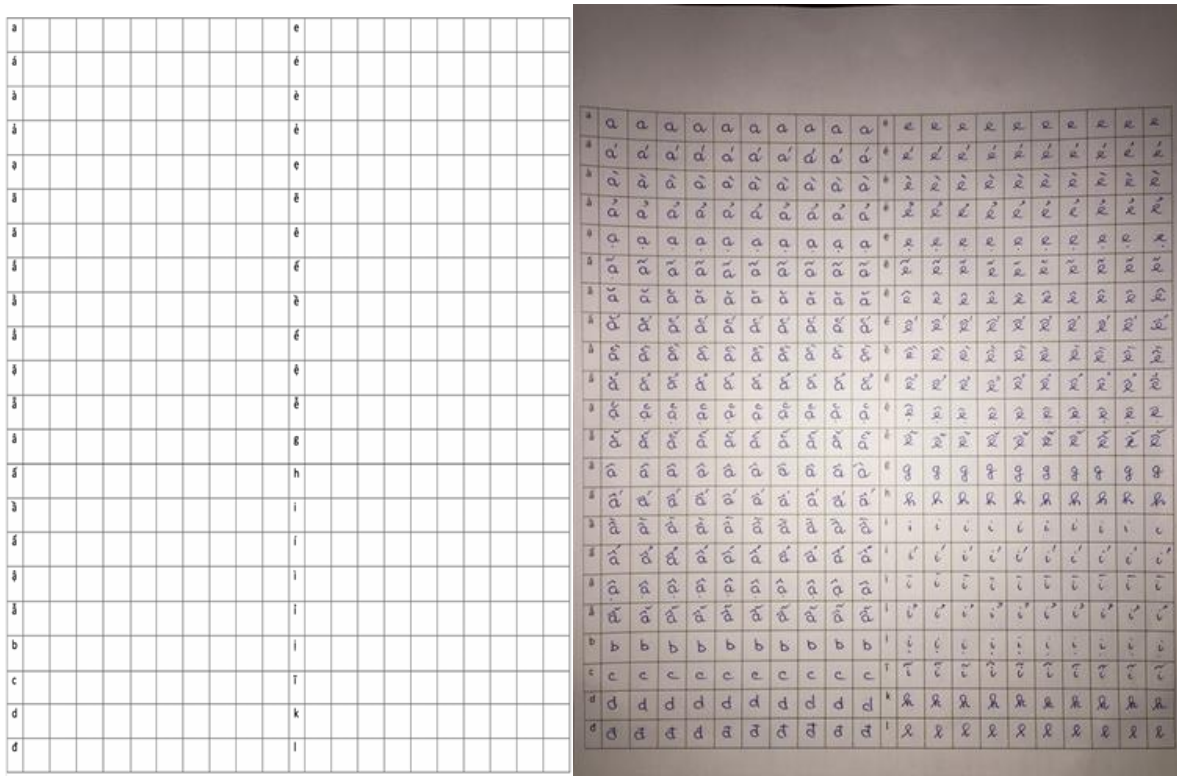
Bộ dữ liệu: Tác giả xây dựng bộ dữ liệu chữ viết tay tiếng Việt (VietData) phục vụ cho việc thực nghiệm bao gồm 89 lớp chữ cái in hoa, mỗi lớp chọn ra 200 mẫu, như vậy bộ dữ liệu VietData có tổng cộng 17800 mẫu. Kết hợp cùng với dữ liệu được thu thập từ hơn 30 người tình nguyện ở thành phố Quảng Ngãi và Quảng Nam.

Nhóm sẽ thu thập thêm dữ liệu để làm tập Test set riêng, với mục đích đánh giá độ chính xác của mô hình.

Quy trình thu thập:

- Chuẩn bị các mẫu giấy A4 (được minh họa trong hình 2)
- Nhờ người viết tay các chữ vào các ô giấy đã chuẩn bị
- Kết quả sau khi viết sẽ như minh họa trong hình 3

Đây là một phương pháp phổ biến trong việc thu thập dữ liệu chữ viết tay để tạo tập test set độc lập, giúp đánh giá khách quan hiệu suất của mô hình máy học.



Hình 22. Mẫu dữ liệu đã được thu thập

Hình 23. Phiếu mẫu để thu thập dữ liệu

Bước 1: Sử dụng cv2 edge detection để cắt gọn những khoảng trắng dư thừa để thuận tiện trong việc lọc các ô chữ

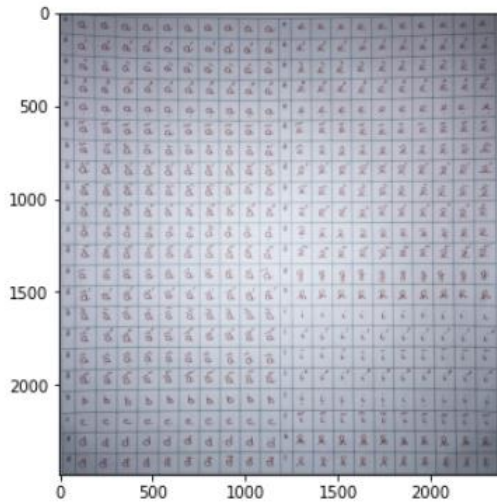
✦ Cắt gọn những khoảng trắng dư thừa

```
def Cut(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 57, 5)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
    max = -1
    L = []
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt) > max:
            x_max, y_max, w_max, h_max = x, y, w, h
            max = cv2.contourArea(cnt)
    table = image[y_max:y_max+h_max, x_max:x_max+w_max]
    return table
```

- Đầu tiên dùng hàm cv2.findContours để lấy được danh sách các contours có trong bức ảnh nhị phân.
- Sau đó dùng hàm cv2.boudingRect để lấy được toạ độ các hình chữ nhật có trong hình. Trong đó x, y là toạ độ trọng tâm của hình chữ nhật và w, h là chiều dài và chiều rộng của hình chữ nhật đó.
- Sau khi tìm được toạ độ của hình chữ nhật lớn nhất, nhóm thực hiện cắt và được hình như bên dưới.

```
# Hình sau khi được cắt gọn các khoảng trắng
image = Cut(image)
plt.figure(figsize=(5,10))
plt.imshow(image, cmap='gray')
```

<matplotlib.image.AxesImage at 0x7f2ec8217a90>



Bước 2: Lọc từng ô chữ sau khi đã được cắt gọn

```
cnts = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
for c in cnts:
    area = cv2.contourArea(c)
    if area < 1000:
        cv2.drawContours(thresh, [c], -1, (0,0,0), -1)

# Xóa các yếu tố gây nhiễu
vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, vertical_kernel, iterations=9)
horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,1))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, horizontal_kernel, iterations=4)

# Sắp xếp theo hàng trên xuống dưới và từng hàng từ trái sang phải
invert = 255 - thresh
cnts = cv2.findContours(invert, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
(cnts, _) = contours.sort_contours(cnts, method="top-to-bottom")

data_rows = []
row = []
for (i, c) in enumerate(cnts, 1):
    area = cv2.contourArea(c)
    if area < 50000:
        row.append(c)
        if i % 9 == 0:
            (cnts, _) = contours.sort_contours(row, method="left-to-right")
            data_rows.append(cnts)
            row = []
```

Sau khi đã có được vị trí của các hàng và vị trí của các từng ô trong mỗi hàng. Tiến hành duyệt từng ô chữ và lưu vào drive.

```
# Lặp lại từng ô
count = 1
for row in data_rows:
    for c in row:
        mask = np.zeros(image.shape, dtype=np.uint8)
        cv2.drawContours(mask, [c], -1, (255,255,255), -1)
        result = cv2.bitwise_and(image, mask)
        result[result==0] = 255
        img_result = result
        try:
            final = Cut(img_result)
            final = cv2.cvtColor(final, cv2.COLOR_BGR2GRAY)
            final = final[10:110, 10:110]
            final = cv2.cvtColor(final, cv2.COLOR_GRAY2RGB)
            cv2.imwrite('/content/gdrive/My Drive/Data_Final_Project/Data/Test/image' + str(count) + '.JPG', final)
            count += 1
        except:
            continue
```

Bước 3: Sau khi thực hiện việc lọc và cắt từng tấm ảnh chỉ chứa 1 chữ cái riêng biệt sau đó phân loại các tấm ảnh về thành những thư mục riêng.



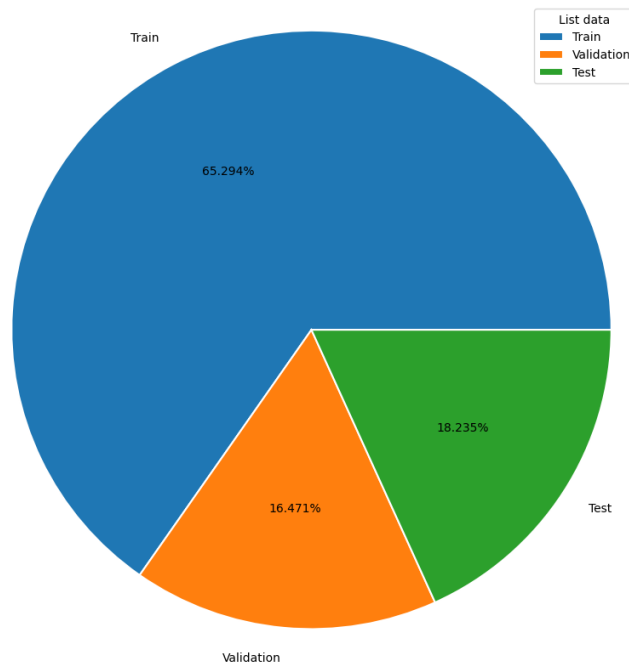
2. Phân chia dữ liệu (Preprocess_and_split_data.ipynb)

Sau khi phân loại và gán nhãn cho dữ liệu, nhóm thống kê được tổng cộng 29.525 mẫu với 89 class, trung bình mỗi class sẽ có khoảng 332 tấm ảnh.

Nhóm chia dữ liệu thu thập được thành 3 tập training set, validation set và test set với số lượng như sau:

- Training set với 19278 mẫu, các mẫu từ training set và validation set được thu thập từ nhiều người viết khác nhau và mỗi người chỉ được viết từ thu thập dữ liệu tối đa là hai lần.

- Validation set với 4.863 mẫu, dùng để đánh giá mô hình sau khi train. Các mẫu ở tập này không được dùng để huấn luyện mô hình.
- Test set với 5384 mẫu được thu thập riêng biệt với hai tập trên.



Hình 24. PieChart

Chương 4. Xử lý dữ liệu và áp dụng các phương pháp trích xuất trung

🔥 Mục tiêu của chương là xử lý dữ liệu chữ viết tay tiếng Việt bằng cách chuyển đổi ảnh sang dạng nhị phân, loại bỏ khoảng trắng, chuẩn hóa kích thước và trích xuất đặc trưng bằng HOG. Sau đó, áp dụng ba mô hình học máy: Logistic Regression, SVM, và MLP để so sánh hiệu suất trên các tập dữ liệu. Các tham số của mô hình được tinh chỉnh để tối ưu hóa kết quả, với SVM kernel RBF đạt độ chính xác cao nhất. Chương này xác định mô hình hiệu quả nhất cho bài toán phân loại chữ viết tay.

❖ Tiền xử lý dữ liệu:

- Chuyển đổi ảnh sang dạng nhị phân (trắng đen).
- Loại bỏ khoảng trắng dư thừa và chuẩn hóa kích thước ảnh về 16x16 pixel.

❖ Trích xuất đặc trưng bằng HOG

- Tính độ lớn và phương gradient từ ảnh.

- Sử dụng histogram để tạo vector đặc trưng HOG đại diện cho mỗi ảnh.
- Vector HOG cuối cùng có 1512 phần tử, đại diện cho toàn bộ ảnh.
- ❖ **Cài đặt và thử nghiệm mô hình**
 - Áp dụng và so sánh ba mô hình học máy: Logistic Regression, SVM, và MLP.
 - Sử dụng các kernel khác nhau cho SVM (linear, polynomial, rbf).
- ❖ **Tinh chỉnh tham số mô hình**
 - Điều chỉnh các tham số như C, max_iter, gamma, và hidden_layer_sizes để tối ưu hiệu suất.
- ❖ **Kết quả và đánh giá**
 - SVM với kernel RBF đạt độ chính xác cao nhất trên tập validation và test.
 - Lựa chọn mô hình phù hợp nhất cho bài toán phân loại chữ viết tay.

1. Tiền xử lý dữ liệu (Preproces_and_split_data)

Mỗi tấm ảnh trong tập train và validation đều được chuyển thành ảnh nhị phân (trắng đen) và xử lý nhiễu. Sau đó tiến hành cắt bớt các khoảng trắng dư thừa xung quanh chữ. Trả về phần ảnh được cắt từ ảnh gốc sử dụng tọa độ đã tìm được vùng cắt sẽ chứa tất cả các đối tượng được phát hiện trong ảnh

```
[ ] # Hàm cắt gọn ảnh bằng cách xác định các contours
def crop_images(img):
    blur = cv2.GaussianBlur(img,(7,7),0)
    thresh = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,7,7)
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]
    x_min = 10**9
    x_max = 0
    y_min = 10**9
    y_max = 0

    for cnt in contours:
        x, y, width, height = cv2.boundingRect(cnt)
        if cv2.contourArea(cnt) > 0:
            x_min = min(x_min, x)
            y_min = min(y_min, y)

            if x + width > x_max:
                x_max = x + width

            if y + height > y_max:
                y_max = y + height

    table = thresh[y_min: y_max, x_min:x_max]
    return table
```

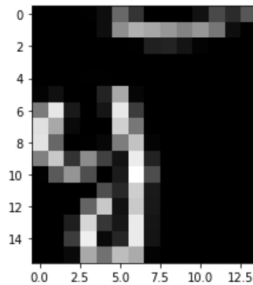
Chuẩn hoá ảnh về kích thước 16x16, đây là kích thước vừa đủ để biểu diễn ảnh ký tự.

Resize tất cả ảnh về cùng kích thước

```
In [ ]: # Test thử 1 ảnh
link_a_img = os.listdir(char_link[85])
img = Image.open(FJoin(char_link[85], link_a_img[30]))
print(img.size)
img = img.resize((16, 16))
plt.imshow(img, cmap = 'gray')
```

(40, 71)

Out[]: <matplotlib.image.AxesImage at 0x7f9f522a0fd0>



2. Áp dụng kỹ thuật HOG để trích xuất đặc trưng (Take_Features_by_HOG)

Từ một bức ảnh ta sẽ lấy ra 2 ma trận quan trọng giúp lưu thông tin ảnh đó là độ lớn gradient (gradient magnitude) và phương của gradient (gradient orientation). Bằng cách kết hợp 2 thông tin này vào một biểu đồ phân phối histogram, cuối cùng chúng ta sẽ thu được một vector mang những đặc trưng HOG (các số thực) đại diện cho một ảnh.

a) Tính Gradient

Trong xử lý ảnh, độ dốc (tức gradient) đang nói đến ở đây chính là độ dốc về mức sáng. Hay nói cách khác chính là sự thay đổi các giá trị pixel trong ảnh. - Tính toán gradient theo trục Ox (Gx) và Oy (Gy) bằng hàm có sẵn trong OpenCV

Kích thước của Gx và Gy đều bằng ảnh gốc. Như vậy mỗi pixel trên Gx sẽ ứng với 1 pixel ở tọa độ tương ứng trên Gy. Vì vậy giá trị độ lớn gradient và phương gradient được tạo ra từ 2 gradient Gx và Gy sẽ xác định được độ lớn và phương tại mỗi pixel.

- Công thức tính độ lớn của gradient:

$$G = \sqrt{G_x^2 + G_y^2}$$

- Công thức tính phương của gradient:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

- Sử dụng hàm có sẵn trong OpenCV để tính độ lớn và phương của gradient:

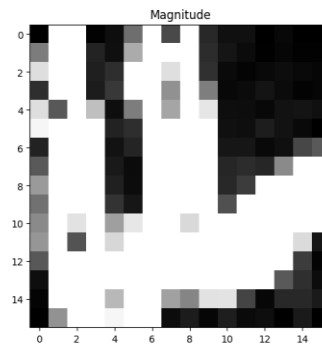
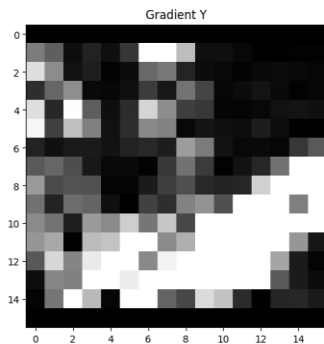
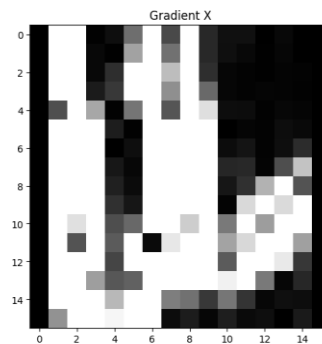
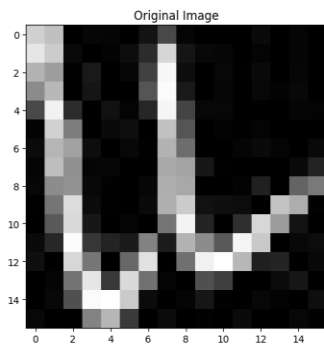
```
import cv2
import numpy as np
# Đọc ảnh
picture_24 = cv2.imread('picture_24.jpg', 0) # đọc ảnh dưới dạng grayscale
# Tính gradient
gx = cv2.Sobel(picture_24, cv2.CV_64F, 1, 0, ksize=3)
gy = cv2.Sobel(picture_24, cv2.CV_64F, 0, 1, ksize=3)
# Tính magnitude và angle
magnitude = np.sqrt(gx**2 + gy**2)
angle = np.arctan2(gy, gx) * 180 / np.pi % 180
# Hiển thị kết quả
# Hiển thị ảnh
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
# Ảnh gốc
plt.subplot(221)
plt.imshow(picture_24, cmap='gray')
plt.title('Original Image')
# Gradient X
plt.subplot(222)
plt.imshow(cv2.convertScaleAbs(gx), cmap='gray')
plt.title('Gradient X')
# Gradient Y
plt.subplot(223)
plt.imshow(cv2.convertScaleAbs(gy), cmap='gray')
plt.title('Gradient Y')
# Magnitude
plt.subplot(224)
plt.imshow(cv2.convertScaleAbs(magnitude), cmap='gray')
plt.title('Magnitude')
plt.tight_layout()
plt.show()
# Đợi phím bất kỳ để đóng cửa sổ
cv2.waitKey(0)
```



```

cv2.destroyAllWindows()
# In thêm thông tin thống kê
# In kích thước của các ma trận
print(f"Kích thước ảnh: {picture_24.shape}")
print(f"Kích thước gradient X: {gx.shape}")
print(f"Kích thước gradient Y: {gy.shape}")
# In giá trị thống kê
print("\nThống kê:")
print(f"Gradient X: min={np.min(gx):.2f}, max={np.max(gx):.2f}")
print(f"Gradient Y: min={np.min(gy):.2f}, max={np.max(gy):.2f}")
print(f"Magnitude: min={np.min(magnitude):.2f}, max={np.max(magnitude):.2f}")
print(f"Angle: min={np.min(angle):.2f}, max={np.max(angle):.2f}")

```

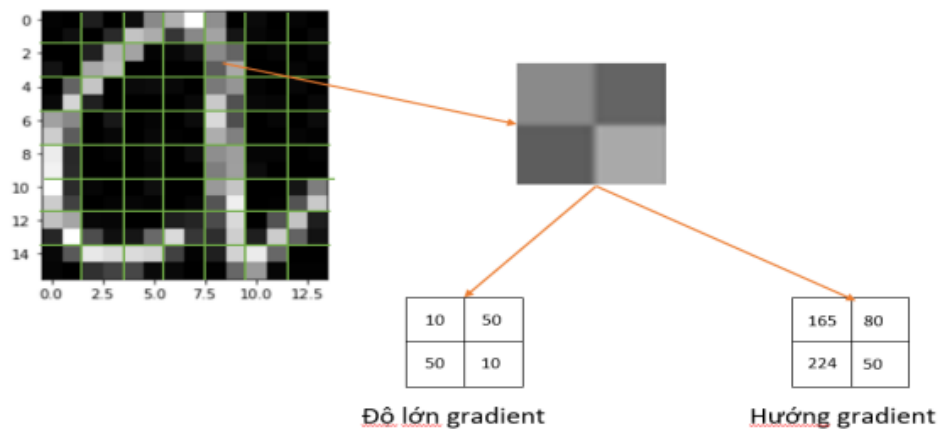


Kích thước ảnh: (16, 16)
 Kích thước gradient X: (16, 16)
 Kích thước gradient Y: (16, 16)

Thống kê:
 Gradient X: min=-987.00, max=984.00
 Gradient Y: min=-905.00, max=849.00
 Magnitude: min=0.00, max=993.68
 Angle: min=0.00, max=179.83

Để làm được như vậy thì hình ảnh được chia thành một lưới ô vuông, mỗi ô vuông có kích thước 2 x 2 pixels. Như vậy ta sẽ có 4 giá trị hướng và 4 giá trị độ lớn ứng với mỗi ô như ma trận bên dưới. - Sau khi tìm được hướng và độ lớn của các khối nhỏ 2 x 2 pixels, ta sẽ vote giá trị độ lớn của mỗi pixel vào khoảng hướng có cùng vị trí tọa độ vào 1 trong 9 bin sau khi xác định được hướng của pixel thuộc

pin tương ứng.



Sau khi tìm được hướng và độ lớn của các khối nhỏ 2 x 2 pixels, ta sẽ gán giá trị độ lớn của mỗi pixel vào khoảng hướng có cùng vị trí tọa độ vào 1 trong 9 bin sau khi xác định được hướng của pixel thuộc pin tương ứng.

- Trong trường hợp độ lớn của phương gradient không chia hết cho đầu mút (nghĩa là không chia hết cho 0, 20, 40...), ta sẽ sử dụng công thức linear interpolation để phân chia độ lớn gradient về 2 biên liền kề.

• **Công thức linear interpolation:**

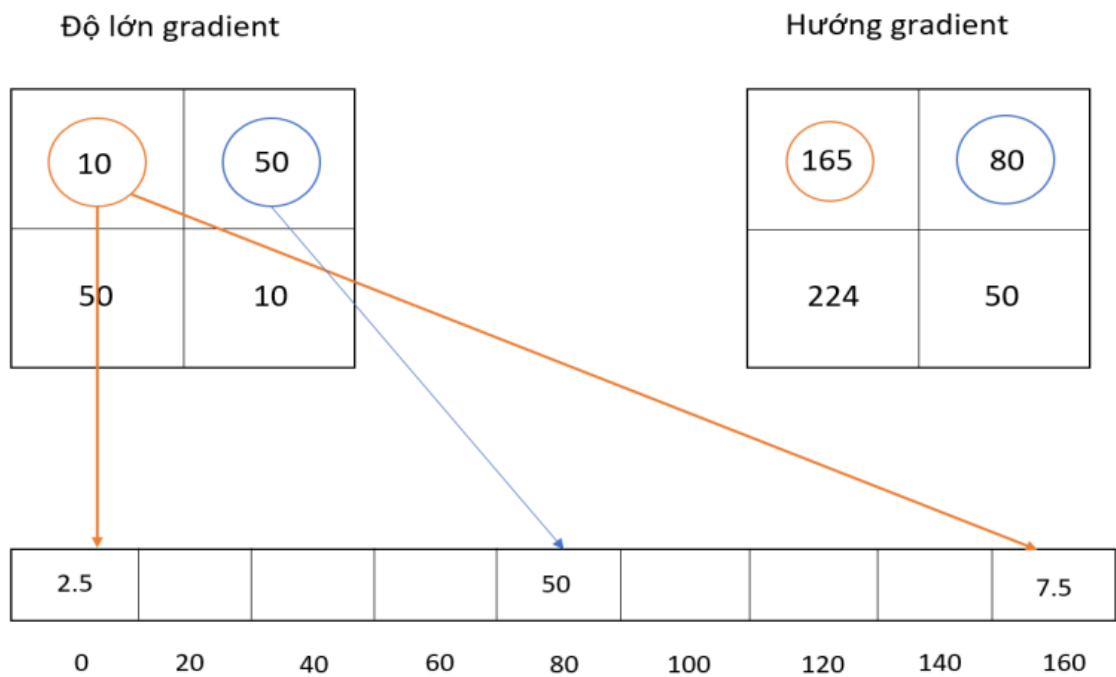
o Giá trị phương gradient bằng x tương ứng với độ lớn gradient y có cùng vị trí tọa độ (x thuộc $[x_1, x_0]$). Khi đó:

$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y$$

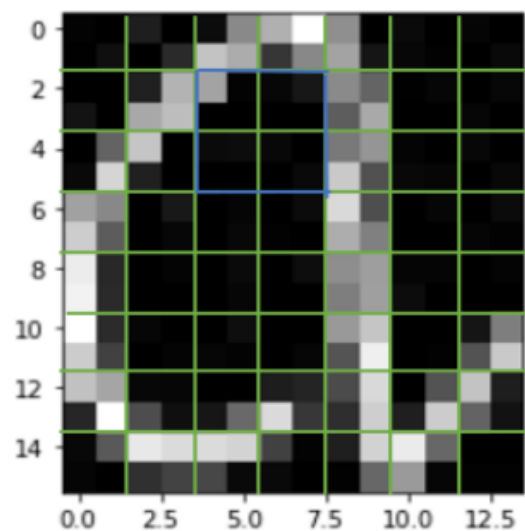
o Tại bin thứ l – 1:

o Tại bin thứ l:

$$x_l = \frac{(x - x_0)}{x_1 - x_0} * y$$



- Tiếp theo, ta sẽ chuẩn hóa vector histogram theo block gồm 4 ô, mỗi ô 2x2 pixel.



Quá trình chuẩn hóa sẽ được thực hiện chuẩn hóa một block có kích thước 2 x 2 đầu tiên, mỗi block chứa 4 ô, mỗi ô có kích thước 2 x 2 pixel, như vậy chúng ta sẽ có 4 vector histogram kích thước 1 x 9 và khi ghép nối tiếp nhau sẽ được một vector có 36 phần tử. Sau đó ta chuẩn hóa theo công thức bên dưới

$$L2\text{-norm}, \mathbf{v} \rightarrow \mathbf{v} / \sqrt{\|\mathbf{v}\|_2^2 + \epsilon^2};$$

L2 – norm: sau chuẩn hóa, độ dài của vector bằng 1.

$\|\mathbf{v}\|_2^2$: Là bình phương của chuẩn L2 của vector \mathbf{v}

ϵ^2 (epsilon bình phương): Là một số dương rất nhỏ (thường là 10^{-5} hoặc 10^{-6})

- Sau đó dịch block đó sang 1 ô và ta sẽ thực hiện chuẩn hóa cho block đó. Đầu vào là một ảnh có kích thước 14x16, áp dụng thuật toán tính HOG với kích thước cells là 2x2 chúng ta sẽ thu được một lưới ô vuông có kích thước $14/2 = 7$ ô theo chiều rộng và $16/2 = 8$ ô theo chiều dài. Sau khi khối block có kích thước 2x2 trải qua 6 bước theo chiều rộng và 7 bước theo chiều và ghép nối tiếp các vector có 36 phần tử lại với nhau ta sẽ có một vector có $36 \times 6 \times 7 = 1512$ phần tử. Đây là **vector HOG** đại diện cho toàn bộ hình ảnh.

Tiến hành trích xuất đặc trưng mẫu 1 hình

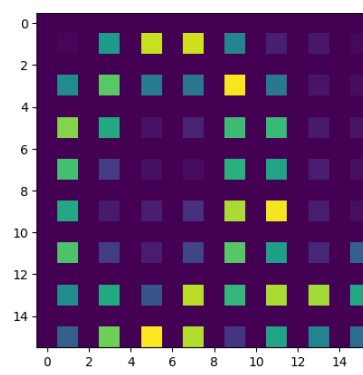
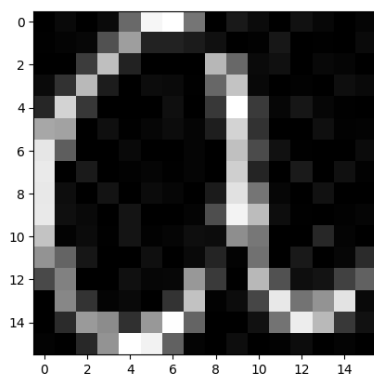
✓ Test trích xuất đặc trưng 1 hình

```

an_img = Image.open('/content/drive/MyDrive/Data_Final_Project/Data/Test/a/picture_107.jpg')
plt.imshow(an_img, cmap = 'gray')
#link_an_img = np.array(link_an_img)
img,im = hog(an_img, pixels_per_cell=(2,2), cells_per_block=(2,2), visualize=True)
plt.figure()
plt.imshow(im)
print(im.size)
print(np.asarray(im).flatten().shape)

```

256
(256,)



Tương tự thực hiện với tập Train, Validation, Test

```
[ ] def getFeature_Label(link, label):
    list_feature = []
    for img_link in link:
        img = Image.open(img_link).convert('L') # Convert to grayscale ('L') instead of 'LA'
        #hàm lấy feature với các ô block để chuẩn hóa có kích thước 2x2 và mỗi ô trong block có kích thước 2x2 pixel
        img = hog(img, pixels_per_cell=(2,2), cells_per_block=(2,2), visualize=False)
        list_feature.append(np.asarray(img).ravel())
        #print(np.asarray(img).ravel().shape)
    return np.array(list_feature), np.array(label)
```

Chương 5. Cài đặt và tinh chỉnh tham số

👉 Mục tiêu của Chương V là cài đặt, thử nghiệm và tinh chỉnh các mô hình học máy gồm Logistic Regression, SVM, và MLP để tối ưu hiệu suất phân loại chữ viết tay tiếng Việt. Chương tập trung vào điều chỉnh các tham số quan trọng như C, max_iter, gamma, kernel và hidden_layer_sizes, sau đó so sánh độ chính xác trên tập validation và test. Kết quả giúp xác định mô hình và tham số tối ưu nhất, đảm bảo hiệu suất cao cho hệ thống phân loại.

❖ Thử nghiệm các mô hình học máy

- Tiến hành huấn luyện và đánh giá ba mô hình: Logistic Regression, SVM, và MLP.

❖ Tinh chỉnh tham số

- Điều chỉnh các tham số quan trọng:
 - Logistic Regression: C, max_iter.
 - SVM: C, gamma, kernel (linear, polynomial, rbf).
 - MLP: hidden_layer_sizes.

❖ Đánh giá hiệu suất mô hình

- So sánh độ chính xác của các mô hình trên tập validation và test.

❖ Lựa chọn mô hình tối ưu

- Xác định mô hình và tham số phù hợp nhất cho bài toán phân loại chữ viết tay tiếng Việt.

1. Thử nghiệm trên Logistic Regression

- Logistic Regression: Là mô hình phân loại nhị phân, dùng để dự đoán xác suất một mẫu thuộc về một lớp cụ thể. Hoạt động bằng cách áp dụng hàm sigmoid lên kết quả của hàm tuyến tính để cho ra xác suất từ 0 đến 1

```
✓ 1m [ ] from sklearn.linear_model import LogisticRegression
model_LG = LogisticRegression(C = 0.1, max_iter=1000)
model_LG.fit(X_train, Y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-v
y = column_or_1d(y, warn=True)

LogisticRegression
LogisticRegression(C=0.1, max_iter=1000)
```

Các thông số:

- C: Thông số dùng để ngăn chặn sự overfit của mô hình, C càng nhỏ thì mô hình sẽ ít bị overfit (nhưng quá nhỏ sẽ bị underfit)
- Max_iter: Số lần lặp tối đa để thuật toán có thể tìm ra local minimum của bài toán.

2. Thực nghiệm trên Support vector machine (SVM)

- Support vector machine (SVM): Là mô hình tìm một siêu phẳng (hyperplane) tối ưu để phân tách các lớp dữ liệu. Hoạt động dựa trên nguyên tắc tối đa hóa khoảng cách (margin) giữa siêu phẳng và các điểm dữ liệu gần nhất. Có thể xử lý dữ liệu phi tuyến tính thông qua kernel trick

Thực hiện training sử dụng model SVM với 3 kernel: linear, polynomial và rbf. Kết quả cho thấy accuracy trên tập validation và tập test nếu sử dụng kernel rbf là cao nhất.

```
[ ] from sklearn.svm import SVC
model_SVM_rbf = SVC(C=1000, kernel = 'rbf', gamma=0.001)
model_SVM_rbf.fit(X_train, Y_train)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.p
y = column_or_1d(y, warn=True)
```

- Các thông số:

- C (Regularization): Tham số điều chỉnh việc có nên bỏ qua các điểm dữ liệu bất thường trong quá trình tối ưu mô hình SVM. Nếu tham số này có giá trị lớn, quá trình tối ưu sẽ chọn một siêu phẳng sao cho siêu phẳng này phân cách tất cả các điểm dữ liệu một cách tốt nhất, từ đó khoảng cách giữa siêu phẳng tới các điểm dữ liệu của các lớp sẽ có giá trị nhỏ (small-margin).
- Gamma: Tham số gamma xác định việc sử dụng bao nhiêu điểm dữ liệu cho việc xây dựng siêu phẳng phân cách. Với giá trị gamma nhỏ, các điểm dữ liệu nằm xa đường phân cách sẽ được sử dụng trong việc tính toán đường phân cách. Ngược lại, với giá trị gamma lớn, chỉ những điểm nằm gần đường phân cách mới được sử dụng để tính toán.

3. Thực nghiệm trên Multi layer Perceptron (MLPClassifier)

- Multi layer Perceptron (MLPClassifier)

Thông số: Hidden_layer_sizes: Số lượng hidden layer và số neuron của mỗi layer.

```
[ ] from sklearn.neural_network import MLPClassifier
```

```
[ ] Y_train = Y_train.ravel()  
    Y_test = Y_test.ravel()  
    Y_vali = Y_vali.ravel()  
    Y_train
```

Chương 6. Training và đánh giá kết quả

👉 Chương VI đánh giá hiệu suất các mô hình Logistic Regression, SVM, và MLP sau khi huấn luyện và kiểm thử trên dữ liệu trích xuất đặc trưng bằng HOG. Kết quả cho thấy MLP đạt độ chính xác cao nhất với hiệu suất test 63%, trong khi Logistic Regression và SVM lần lượt đạt 57% và 60%. Phương pháp HOG chứng minh hiệu quả trong việc cải thiện độ chính xác, tuy nhiên tỷ lệ dự đoán sai vẫn cao trên một số lớp.

❖ Thời gian huấn luyện và kiểm thử

- Logistic Regression: Train 2m 45s, test 42s.
- SVM: Train 21m 10s, test 19m 48s.
- MLP: Train 9m 14s, test 48s.

❖ Đánh giá hiệu suất các mô hình

- Logistic Regression: Validation tăng từ 49% → 70%, test từ 38% → 57%.
- SVM: Validation tăng từ 72% → 74%, test từ 53% → 60%.
- MLP: Validation tăng từ 73% → 77%, test từ 55% → 63%.

❖ Hiệu quả của phương pháp HOG

- HOG cải thiện đáng kể accuracy trên tất cả các mô hình, thể hiện khả năng trích xuất đặc trưng tốt.

❖ Nhận xét chung

- MLP đạt hiệu suất cao nhất trong ba mô hình.

- Tuy nhiên, tỷ lệ dự đoán sai trên một số lớp vẫn cao, cần nghiên cứu thêm các mô hình tiên tiến và tối ưu hóa dữ liệu để cải thiện hiệu suất.

1. Kết quả với HOG

a. Thuật toán Logistic Regression

- Thời gian train: 2m 45s
- Thời gian test (tập validation và tập test): 42s
- Đánh giá kết quả: + Kết quả trên tập validation:
- + Kết quả trên tập validation:

accuracy			0.70	4863
macro avg	0.70	0.68	0.68	4863
weighted avg	0.70	0.70	0.70	4863

- + Kết quả trên tập test:

accuracy			0.57	5384
macro avg	0.58	0.57	0.56	5384
weighted avg	0.58	0.57	0.56	5384

b. Thuật toán SVM

- Thời gian train: 21m 10s
- Thời gian test (tập validation và tập test): 19m 48s
- Đánh giá kết quả:
- + Kết quả trên tập validation:

accuracy			0.74	4863
macro avg	0.73	0.72	0.72	4863
weighted avg	0.74	0.74	0.74	4863

- + Kết quả trên tập test:

accuracy			0.60	5384
macro avg	0.61	0.60	0.59	5384
weighted avg	0.61	0.60	0.59	5384

c. Thuật toán MLPClassifier

Thời gian train: 9m 14s

- Thời gian test (tập validation và tập test): 48s

- Đánh giá kết quả:

+ Kết quả trên tập validation:

accuracy			0.77	4863
macro avg	0.78	0.76	0.76	4863
weighted avg	0.79	0.77	0.77	4863

+ Kết quả trên tập test:

accuracy			0.63	5384
macro avg	0.65	0.63	0.63	5384
weighted avg	0.65	0.63	0.63	5384

2. Nhận xét chung

Đánh giá hiệu suất các mô hình qua chỉ số accuracy:

- Đối với Logistic Regression:
 - Hiệu suất trên tập validation cải thiện đáng kể: 49% → 70%
 - Hiệu suất trên tập test tăng mạnh: 38% → 57%
- Với mô hình SVM:
 - Độ chính xác validation được cải thiện nhẹ: 72% → 74%
 - Độ chính xác test có sự tăng trưởng: 53% → 60%
- Mô hình MLP thể hiện:
 - Hiệu suất validation tăng ổn định: 73% → 77%
 - Hiệu suất test cải thiện tích cực: 55% → 63%

Phương pháp HOG đã chứng minh hiệu quả trong việc trích xuất đặc trưng. Kết quả cho thấy accuracy của tất cả các mô hình đều được cải thiện khi áp dụng HOG. Vì vậy, đề xuất tiếp tục sử dụng HOG cho các bài toán trích xuất đặc trưng ảnh tiếp theo.

Trong ba mô hình được thử nghiệm, MLP thể hiện hiệu suất vượt trội nhất.

Tuy nhiên, khi so sánh với các nghiên cứu trước đây, kết quả accuracy của các mô hình vẫn còn hạn chế. Tỷ lệ dự đoán sai trên một số lớp vẫn cao, cho thấy cần thiết phải:

- Nghiên cứu thêm các mô hình tiên tiến hơn
- Tìm hiểu sâu hơn về cách tối ưu hóa các mô hình hiện tại
- Cải thiện chất lượng dữ liệu huấn luyện

Chương 7. Ứng dụng và cải thiện

1. Ứng dụng

- **Hỗ trợ phân loại chữ viết tay trong giáo dục:** Ứng dụng này có thể tích hợp vào các phần mềm học tập, giúp học sinh cải thiện kỹ năng viết bằng cách phân loại các chữ viết tay có dấu. Ngoài ra, nó còn hỗ trợ chấm điểm tự động cho bài kiểm tra viết tay, đặc biệt trong các môn ngôn ngữ, tiết kiệm thời gian và công sức cho giáo viên.
- **Số hóa tài liệu viết tay:** Hệ thống có thể được sử dụng trong các thư viện và trung tâm lưu trữ để số hóa và phân loại tài liệu viết tay tiếng Việt. Việc chuyển đổi các chữ viết tay có dấu sang dạng văn bản số giúp giảm công sức, đồng thời tăng khả năng lưu trữ và tìm kiếm tài liệu.
- **Tích hợp trong thiết bị đầu vào thông minh:** Các thiết bị như bút điện tử (smart pen) hoặc tablet có thể tích hợp tính năng này để phân loại chữ viết tay tiếng Việt có dấu thành văn bản số trong thời gian thực, hỗ trợ người dùng ghi chú và nhập liệu dễ dàng hơn.
- **Hỗ trợ người khuyết tật:** Ứng dụng này có thể giúp người khuyết tật về vận động hoặc thị giác giao tiếp tốt hơn bằng cách chuyển chữ viết tay có dấu thành văn bản số hóa, hỗ trợ trong các tình huống giao tiếp hàng ngày hoặc lưu trữ thông tin cá nhân.
- **Nhận dạng chữ viết trong ngân hàng và tài chính:** Ứng dụng này sẽ hỗ trợ nhận dạng chữ ký hoặc các biểu mẫu viết tay có dấu trong ngân hàng và các tổ chức tài chính, giúp giảm thiểu sai sót và tăng tốc độ xử lý dữ liệu như phiếu ghi nợ, phiếu thanh toán.
- **Phân tích chữ viết trong lĩnh vực pháp lý:** Trong các tình huống pháp lý, ứng dụng có thể được sử dụng để phân tích, nhận dạng và phân loại chữ viết tay có dấu, giúp xử lý các tài liệu pháp lý hoặc hồ sơ điều tra nhanh chóng. Ngoài ra, nó còn hỗ trợ phát hiện giả mạo chữ viết tay.
- **Phát triển chatbot hoặc trợ lý ảo thông minh:** Hệ thống có thể được tích hợp vào chatbot hoặc trợ lý ảo, cho phép người dùng gửi thông tin dưới dạng chữ viết tay có dấu và nhận lại phản hồi nhanh chóng, đảm bảo độ chính xác và trải nghiệm tự nhiên hơn.
- **Ứng dụng trong y tế:** Trong y tế, ứng dụng này có thể giúp nhận dạng và số hóa các đơn thuốc hoặc hồ sơ bệnh án viết tay có dấu, giảm thiểu sai sót và tăng độ chính xác trong quá trình nhập liệu.

- **Dịch thuật tự động:** Bằng cách nhận dạng và chuyển đổi chữ viết tay tiếng Việt có dấu thành văn bản số, hệ thống có thể được tích hợp vào các công cụ dịch thuật tự động để dịch sang các ngôn ngữ khác, hỗ trợ giao tiếp đa ngôn ngữ hiệu quả.
- **Nghiên cứu và bảo tồn văn hóa:** Ứng dụng có thể phân tích, lưu trữ chữ viết tay tiếng Việt có dấu, phục vụ nghiên cứu lịch sử, ngôn ngữ và văn hóa. Đồng thời, nó còn hỗ trợ số hóa các tài liệu viết tay quý giá như nhật ký, thư từ của các danh nhân, góp phần bảo tồn văn hóa dân tộc.

2. Hướng cải thiện

- Để nâng cao chất lượng và hiệu quả của đề án, một số hướng cải thiện có thể được thực hiện như sau: **Cải thiện dữ liệu đầu vào** bằng cách nâng cao chất lượng ảnh chữ viết tay thông qua các phương pháp xử lý ảnh như lọc nhiễu, tăng độ tương phản và chuẩn hóa kích thước. Ngoài ra, tăng số lượng mẫu dữ liệu bằng cách thu thập từ nhiều nguồn đa dạng và loại bỏ các yếu tố nhiễu trong dữ liệu cũng là giải pháp quan trọng.
- **Thử nghiệm các mô hình học sâu tiên tiến** như Convolutional Neural Network (CNN) để trích xuất đặc trưng tự động hoặc sử dụng các mô hình Transformer-based (như Vision Transformer) nhằm cải thiện độ chính xác nhờ khả năng học phụ thuộc dài hạn. Đồng thời, việc áp dụng các kỹ thuật tăng cường dữ liệu như xoay ảnh, thay đổi độ sáng hoặc thêm nhiễu cũng giúp cải thiện khả năng tổng quát hóa của mô hình.
- Ngoài ra, có thể **áp dụng các kỹ thuật trích xuất đặc trưng mạnh mẽ** như SIFT (Scale-Invariant Feature Transform) và SURF (Speeded-Up Robust Features) để cải thiện việc nhận dạng đặc trưng hình ảnh. Kết hợp với đó, sử dụng các phương pháp ensemble để tận dụng điểm mạnh của nhiều mô hình khác nhau, giúp tăng hiệu suất tổng thể.
- Một hướng mở rộng thú vị là **mở rộng bài toán từ phân loại ký tự sang nhận dạng nguyên từ hoặc câu**, điều này sẽ gia tăng khả năng ứng dụng thực tế trong việc chuyển đổi các văn bản viết tay. Đồng thời, kết hợp các mô hình ngôn ngữ như RNN hoặc Transformer để tận dụng ngữ cảnh khi dự đoán từ hoặc câu cũng là một giải pháp cần thử nghiệm.
- Cuối cùng, để ứng dụng thực tế dễ dàng hơn, cần nâng cao khả năng tổng quát hóa của mô hình thông qua việc huấn luyện trên tập dữ liệu đa ngôn ngữ và đánh giá trên các tập dữ liệu thực tế, chẳng hạn như biểu mẫu hoặc ghi chú viết tay. Xây dựng **giao diện người dùng thân thiện** và tích hợp ứng dụng trên các nền tảng web hoặc di động sẽ giúp đề án có giá trị ứng dụng cao hơn trong thực tiễn.

Danh mục tài liệu tham khảo

- (n.d.). From <https://www.noron.vn/post/gioi-thieu-ve-support-vector-machine-trong-machinelearning-40dxtjcmrdye>
- Khoa học dữ liệu*. (2019, November 22). Retrieved January 9, 2025 from Khoa học dữ liệu: <https://phamdinhhkhanh.github.io/2019/11/22/HOG.html>
- Machine Learning cơ bản*. (2017, February 24). Retrieved January 2, 2025 from Machine Learning cơ bản: <https://machinelearningcoban.com/2017/02/24/mlp/>
- NGHIÊN CỨU ỨNG DỤNG PHƯƠNG PHÁP MÁY VÉC TƠ TỰA TRONG NHẬN DẠNG CHỮ VIỆT VIẾT TAY RỒI RẠC*. (n.d.). Retrieved January 3, 2025 from Dữ liệu mở: https://dulieu.itrithuc.vn/media/dataset/2020_08/la10.0233.3.pdf
- Scikit-learn*. (n.d.). From Support Vector Machines: <https://scikit-learn.org/stable/modules/svm.html#svm-kernels>

SOURCE

<https://drive.google.com/drive/folders/1mqzwJcXEKzBvXGuUVs7soJ9OVMLvpdW3?usp=sharing>