HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

HUỲNH MINH NHỰT

NGHIÊN CỨU MÔ HÌNH NHẬN DẠNG CHỮ KÝ VIẾT TAY SỬ DỤNG HỌC SÂU CNN

Chuyên ngành: HỆ THỐNG THÔNG TIN
Mã số: 8.48.01.04

TÓM TẮT LUẬN VĂN THẠC SĨ
(Theo định hướng ứng dụng)

TP.HCM – NĂM 2022

Luận văn được hoàn thành tại:

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Người hướng dẫn khoa học: **TS. NGUYỄN XUÂN SÂM**

Phản biện 1: .....

Phản biện 2: .....

Luận văn sẽ được bảo vệ trước Hội đồng chấm luận văn tại Học viện Công nghệ Bưu chính Viễn Thông

Vào lúc: .....giờ ..... ngày ..... tháng ..... năm .....

Có thể tìm hiểu luận văn tại:

- Thư viện của Học viện Công nghệ Bưu Chính Viễn Thông.

MỞ ĐẦU

Trong lĩnh vực sinh trắc học hành vi, việc xác minh chữ ký là quy trình tham chiếu để xác thực một người. Chữ ký được coi là "con dấu phê duyệt" để xác minh sự chấp thuận của người dùng và vẫn là phương tiện xác thực được ưa chuộng nhất. Chữ ký viết tay vốn áp đặt trách nhiệm pháp lý về tài chính và đạo đức, là một kỹ thuật xác thực vẫn được sử dụng rộng rãi ngày nay, đặc biệt trong các văn bản pháp lý, giao dịch ngân hàng và thương mại. Do đó, chữ ký viết tay thường bị các kẻ xấu lợi dụng và sử dụng để lừa đảo.

Chữ ký viết tay chỉ bao gồm hình dạng của chữ ký. Điều này khiến chúng trở thành một vấn đề đầy thách thức đối với các nhà khoa học.

Ngoài phần mở đầu, mục lục, kết luận và tài liệu tham khảo, nội dung chính của luận án được chia thành 3 chương, cụ thể như sau:

Chương 1: Tổng quan về xây dựng hệ thống.

Chương 2: Cơ sở lý thuyết và các công trình liên quan.

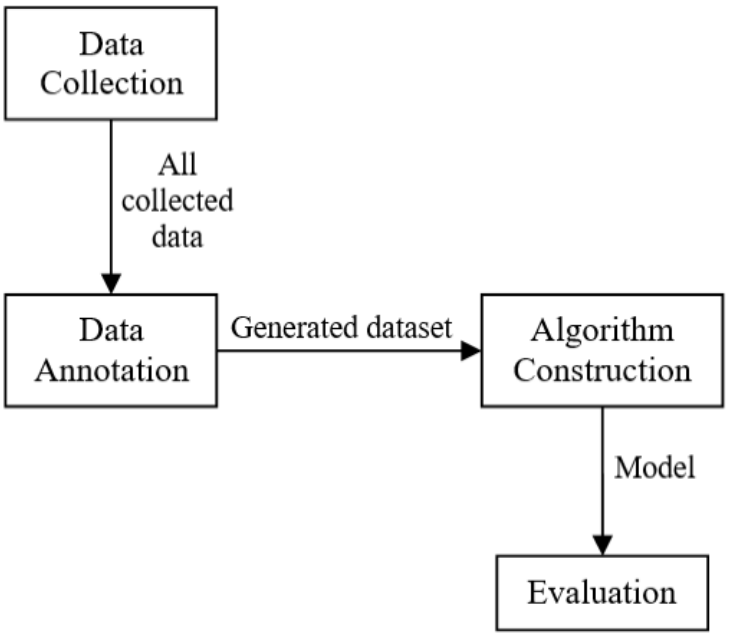
Chương 3: Mô hình nhận biết chữ ký viết tay.

CHƯƠNG 1: TỔNG QUAN VỀ XÂY DỰNG HỆ THỐNG NHẬN DẠNG CHỮ KÝ VIẾT TAY

1.1 Các phương pháp nhận dạng chữ ký viết tay truyền thống

Chữ ký viết tay là một biểu tượng viết tay của con người. Chữ ký [1] còn là một công cụ phổ biến để xác định danh tính của một người, đặc biệt là về mặt phê chuẩn các tài liệu mật. Điều này dẫn đến nhu cầu nghiên cứu thêm về xác minh chữ ký, nhằm ngăn chặn việc các bên thiếu trách nhiệm lợi dụng chữ ký.

Tuy nhiên để xác minh chữ ký thường thực hiện theo 4 giai đoạn sau: thu thập dữ liệu, chú thích dữ liệu, xây dựng mô hình và đánh giá mô hình [3]:

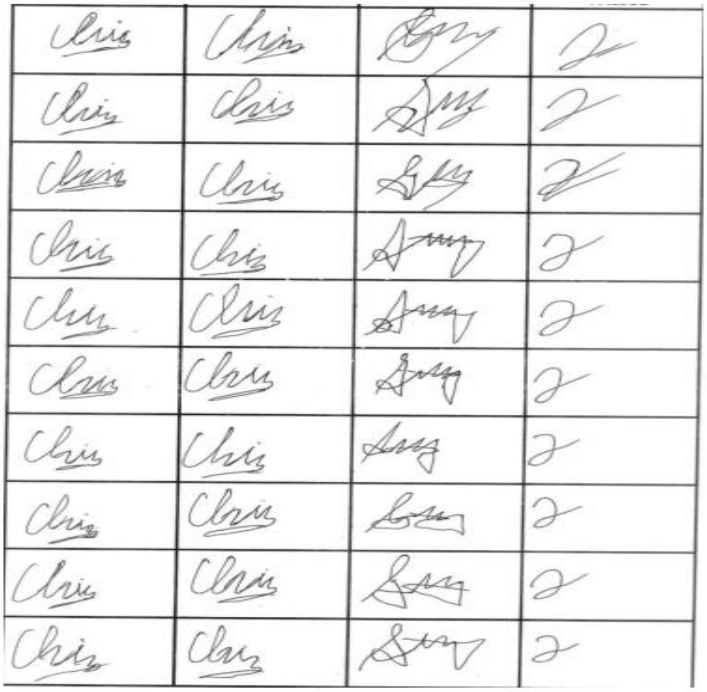


Hình 1.1: Tổng quan về Quy trình Nghiên cứu

Quy trình nghiên cứu của chúng tôi gồm bốn thành phần chính, các thành phần được giải thích chi tiết trong phần này:

Thu thập dữ liệu:

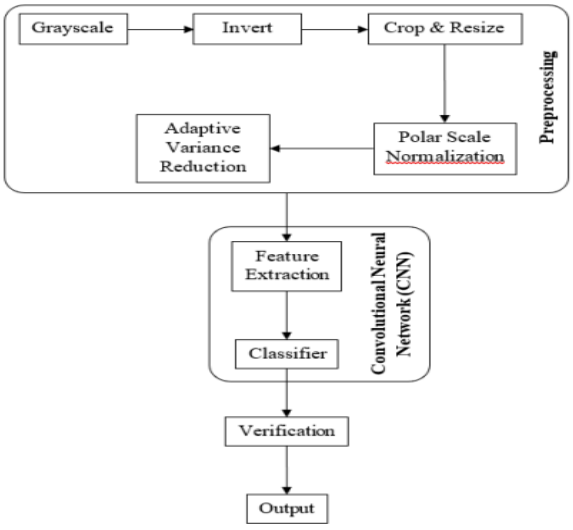
Là tập hợp các quy trình phân tích thu thập dữ liệu của người dùng.



Hình 1.2: Hình ảnh chữ ký trong quá trình thu thập

Ví dụ về dữ liệu được thu thập, hai Cột đầu tiên từ bên trái là chữ ký ban đầu, sau đó là cột thứ ba và thứ tư là chữ ký bắt chước của người khác mà tình nguyện viên đã giả mạo.

Xây dựng mô hình:



Hình 1.3: Tổng quan về xây dựng thuật toán xác minh chữ ký

Đây là giai đoạn quan trọng, sau khi tập dữ liệu thông qua quá trình tiền xử lý, dữ liệu được sử dụng trực tiếp làm đầu vào cho CNN sẽ được xử lý để trích xuất các tính năng. Sau đó, các đặc điểm đã được trích xuất có thể được gửi đến bộ phân loại và quá trình xác minh sẽ tạo ra mô hình được sử dụng để xác định cho dù chữ ký là thật hay giả.

Đánh giá mô hình:

Giai đoạn này là giai đoạn cuối cùng trong quá trình triển khai mô hình. Tại giai đoạn này, mỗi mô hình sẽ được đánh giá, sau đó sẽ tính độ chính xác và tỷ lệ lỗi với một số cấu hình. Để biết chi tiết về sự sắp xếp và kết quả của các cuộc đánh giá.

1.2 Machine Learning trong nhận dạng chữ ký viết tay

1.2.1 Giới thiệu về machine learning

Machine Learning [4],[5] là việc sử dụng và phát triển các hệ thống máy tính có khả năng học hỏi và thích ứng mà không cần tuân theo các chỉ dẫn rõ ràng, bằng các sử dụng các thuật toán và mô hình thống kê để phân tích và rút ra suy luận từ các mẫu trong dữ liệu đầu vào.

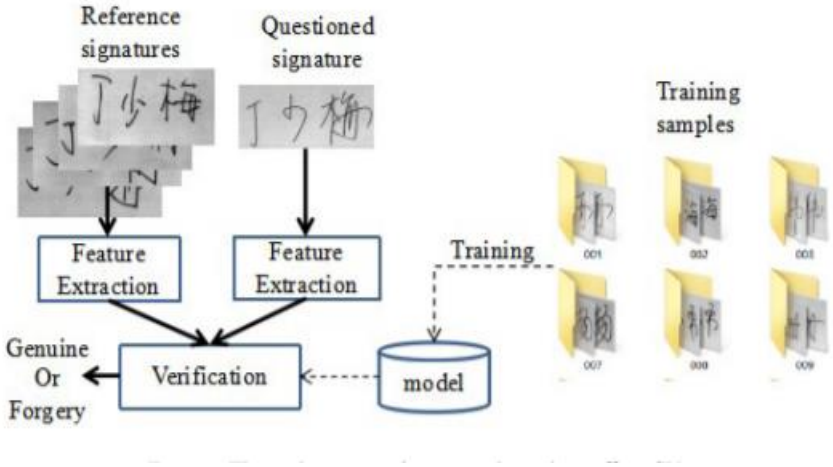
Xác minh chữ ký [6] trực quan được xây dựng một cách tự nhiên như một nhiệm vụ học máy. Một chương trình được cho là thể hiện khả năng máy học trong việc thực hiện một nhiệm vụ nếu nó có thể học hỏi từ những người mẫu, cải thiện khi số lượng người mẫu tăng lên, v.v. [3].

1.2.2 Input và Ouput

Machine Learning sử dụng trong nhận dạng chữ ký viết tay trước đây chủ yếu sử dụng cây quyết định (Dessection Tree) hoặc cao hơn là sử dụng random forest để nhận dạng chúng.

Input: Là các hình ảnh chữ ký được gắn nhãn.

Output: Đưa ra dự đoán về chữ ký của ai trong tập traning ở đây là các nhãn 001, 002, 003?

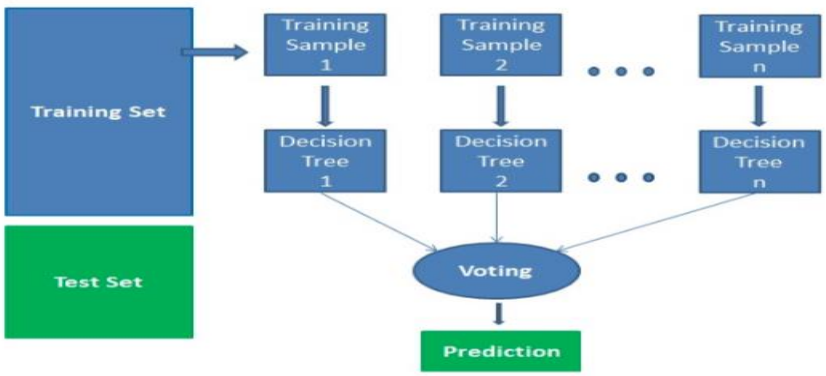


Hình 1.4: Kiến trúc của 1 mô hình nhận dạng chữ ký [1]

1.2.3 Random forest trong nhận dạng chữ ký viết tay

Thuật toán random forest trong nhận dạng chữ ký viết tay sẽ hoạt động theo 5 bước:

- Bước 1: Tiền xử lý tình ảnh
- Bước 2: Chọn các mẫu chữ ký ngẫu nhiên từ tập dữ liệu huấn luyện
- Bước 3: Xây dựng cây quyết định cho mẫu chữ ký
- Bước 4: Các cây quyết định đưa ra các dự đoán
- Bước 5: Chọn kết quả là chữ ký được dự đoán nhiều nhất là dự đoán cuối cùng



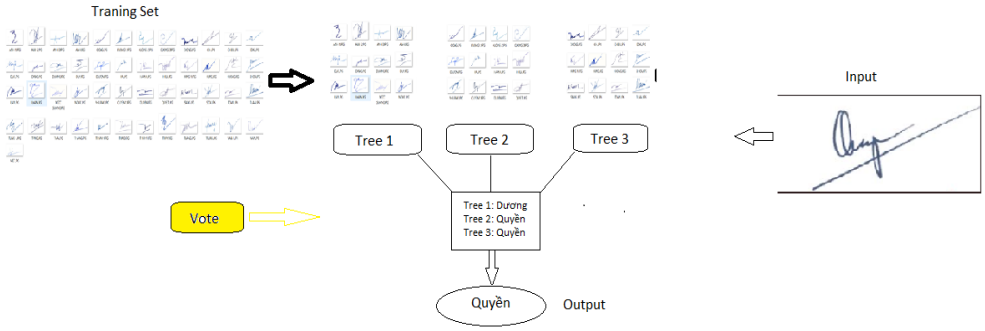
Hình 1.5: Mô tả thuật thoán random forest

Training Set: Là tập các hình ảnh nằm trong các thư mục có gắn nhãn hoặc các hình ảnh có nhãn.

Hình 1.6: Mẫu training set trong random forest

Training Sample: Là các tập được lựa chọn ngẫu nhiên từ tập training set từ đó tạo thành các cây quyết định chuẩn bị cho quá trình voting.

Test: Đưa dữ liệu là hình ảnh chữ ký vào mô hình. Kết thúc quá trình voting sẽ là kết quả của dự đoán



Hình 1.7: Áp dụng random forest vào nhận dạng chữ ký

Ưu điểm:

- Thuật toán mang tính chất mô phỏng dễ hình dung, không phức tạp nhiều về thuật toán.
- Thuật toán được hỗ trợ trên nhiều thuật toán, trên nhiều lĩnh vực khác nhau.

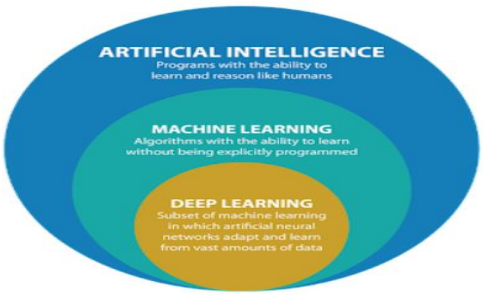
Nhược điểm:

- Độ chính xác không cao.

1.3 Deep Learning (DL) trong nhận dạng chữ ký viết tay

1.3.1 Giới thiệu về deep learning

Hiện nay, AI [7] đang phát triển mạnh mẽ, để đạt được bước tiến như hiện tại thì deep learning như là một chìa khóa thúc đẩy AI ngày càng tiến xa hơn và sử dụng rộng rãi với đời sống con người hơn.

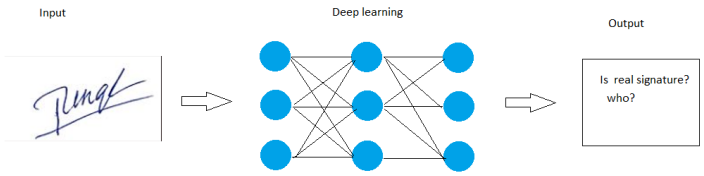


Hình 1.8: Mối liên hệ AI, ML, DL

1.3.2 Deep learning trong xây dựng hệ thống nhận dạng chữ ký viết tay

Bài toán đặt ra: Chúng ta sẽ có 1 bộ các chữ ký thật, giả có gắn nhãn là tên của các nhân vật đại diện cho từng chữ ký. Bài toán đặt ra là khi đưa ảnh của 1 chữ ký vào có thể nhận dạng đó là chữ ký thật hay giả? Và chữ ký đó là của ai?

Để giải quyết vấn đề trên được triệt để đưa ra các dự đoán có độ chính xác cao, có độ tin cậy và an toàn hơn. Trong việc xây dựng này, đề xuất thực hiện bằng deep learning, sử dụng CNN hình thành nên các mạng nơ-ron nhân tạo, tiến hành phân tích dự đoán.



Hình 2.9: Deep learning trong nhận dạng chữ ký viết tay

1.4 Kết luận chương

Hiểu biết được những khái niệm tổng quan về trí tuệ nhân tạo. Thấy được tầm quan trọng và ý nghĩa của trí tuệ nhân tạo. Từ đó, có cái nhìn về học máy, học sâu các lợi ích về sau mà trí tuệ nhân tạo sẽ mang lại cho con người. Áp dụng vào việc sử dụng để nhận dạng chữ ký viết tay.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ CÁC CÔNG TRÌNH LIÊN QUAN

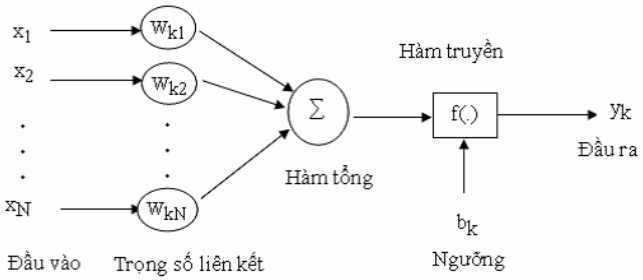
2.1 Cơ sở lý thuyết

2.1.1 Mạng nơ-ron (neural)

2.1.1.1 Giới thiệu về mạng nơ-ron

Mạng nơ-ron nhân tạo (ANN) bao gồm một lớp nút, chứa một lớp đầu vào, một hoặc nhiều lớp ẩn và một lớp đầu ra. Mỗi nút, hoặc nơ-ron nhân tạo, kết nối với một nút khác và có trọng số và ngưỡng liên quan. Nếu đầu ra của bất kỳ nút riêng lẻ nào vượt quá giá trị ngưỡng được chỉ định, nút đó sẽ được kích hoạt, gửi dữ liệu đến lớp tiếp theo của mạng. Nếu không, không có dữ liệu nào được chuyển đến lớp tiếp theo của mạng.

Cấu trúc nơ-ron nhân tạo:



Hình 2.1: Cấu tạo một nơ-ron

Các thành phần cơ bản của một nơ-ron nhân tạo bao gồm:

- Input signals: Là các tín hiệu đầu vào của nơ-ron, các tín hiệu này thường được đưa vào dưới dạng một vector N chiều.
- Thành phần liên kết: được kết nối bởi trọng số liên kết – synaptic weight. Các trọng số kết nối giữa tín hiệu vào thứ j với nơ-ron k thường được kí hiệu là w_{kj}. Thông thường, các trọng số liên kết này được khởi tạo một cách ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục.
- Thành phần tổng (Summing function): được dùng để tính tổng của tích các đầu vào với trọng số liên kết của nó.

- Thành phần ngưỡng (còn gọi là một độ lệch - bias): thành phần này thường được đưa vào như một thành phần của hàm truyền.
- Thành phần truyền (Transfer function): Thành phần này được dùng để giới hạn phạm vi đầu ra của mỗi nơ-ron. Nó nhận đầu vào là kết quả của hàm tổng và ngưỡng.
- Đầu ra: Là tín hiệu đầu ra của một nơ-ron, với mỗi nơ-ron sẽ có tối đa là một đầu ra.

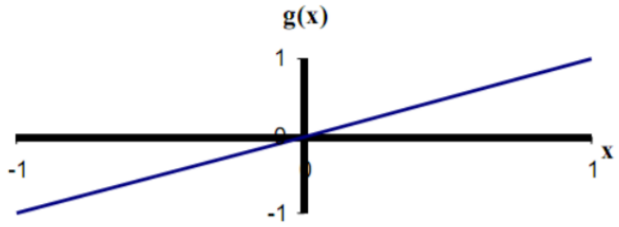
2.1.1.2 Một số hàm truyền thông dụng

Phần lớn các đơn vị trong mạng nơ-ron chuyển net input bằng cách sử dụng một hàm vô hướng (scalar-to-scalar function) [9] gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị (unit's activation). Loại trừ khả năng đơn vị đó thuộc lớp ra, giá trị kích hoạt được đưa vào một hay nhiều đơn vị khác. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp (squashing). Các hàm kích hoạt hay được sử dụng là:

Hàm đồng nhất (Linear function, Identity function)

$$g(x) = x$$

Nếu coi các đầu vào là một đơn vị thì chúng sẽ sử dụng hàm này. Đôi khi một hằng số được nhân với net-input để tạo ra một hàm đồng nhất.



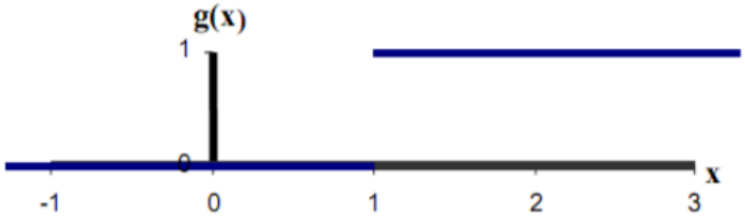
Hình 2.2: Hàm đồng nhất (Identify function)

Hàm bước nhị phân (Binary step function, Hard limit function)

Hàm này cũng được biết đến với tên "Hàm ngưỡng" (Threshold function hay Heaviside function). Đầu ra của hàm này được giới hạn vào một trong hai giá trị:

$$g(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Hàm này được sử dụng một lớp. Trong hình vẽ sau, θ được chọn bằng 1.

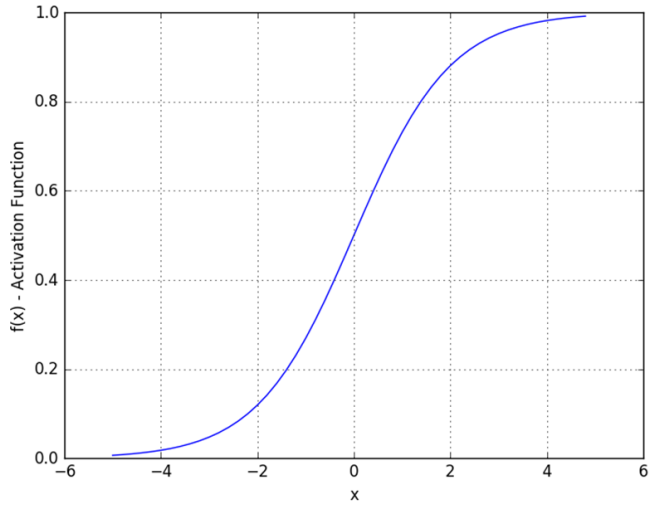


Hình 2.3: Hàm bước nhị phân

Hàm sigmoid (Sigmoid function (logsig))

$$g(x) = \frac{1}{1+e^{-x}}$$

Hàm này rất thuận lợi khi sử dụng vì được huấn luyện bởi thuật toán Lan truyền ngược, nó dễ lấy đạo hàm. Hàm này được ứng dụng cho các chương trình ứng dụng mà các đầu ra mong muốn rơi vào khoảng [0,1].

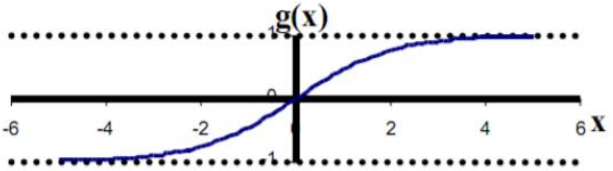


Hình 2.4: Hàm Sigmoid

Hàm sigmoid lưỡng cực (Bipolar sigmoid function (tansig)).

$$g(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Hàm này có các thuộc tính tương tự hàm sigmoid. Nó làm việc tốt đối với các ứng dụng có đầu ra yêu cầu trong khoảng [-1,1].



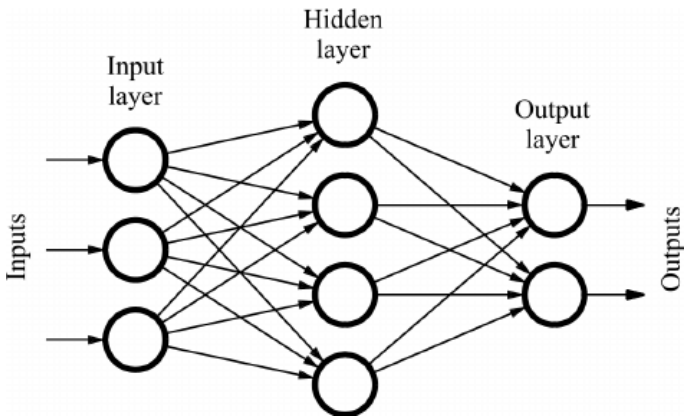
Hình 2.5: Hàm sigmoid lưỡng cực

Hàm này rất thuận lợi khi sử dụng vì được huấn luyện bởi thuật toán Lan truyền ngược, nó dễ lấy đạo hàm. Hàm này được ứng dụng cho các chương trình ứng dụng mà các đầu ra mong muốn rơi vào khoảng [0,1].

2.1.2 Một số kiểu mạng nơ-ron

2.1.2.1 Mạng truyền thẳng (Feed-forward neural network)

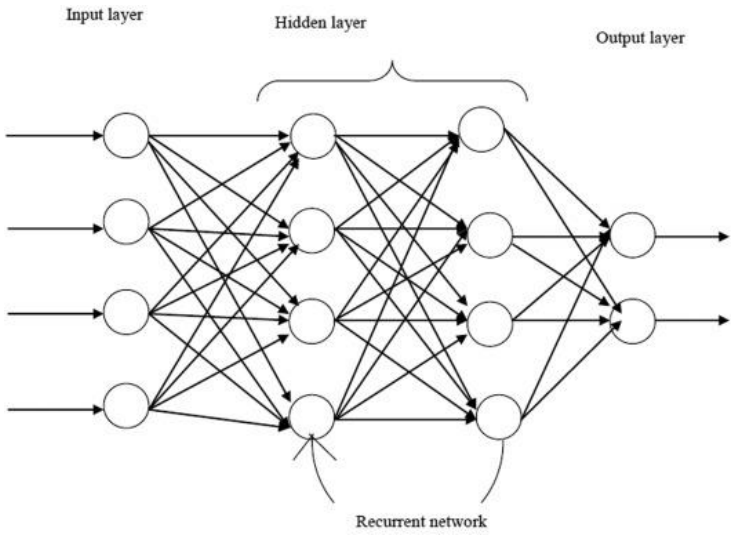
Mạng nơ-ron truyền thẳng là một mạng nơron nhân tạo trong đó các kết nối giữa các nút không tạo thành một chu trình. Đối lập với mạng nơ-ron truyền thẳng là mạng nơ-ron tuần hoàn, trong đó các đường dẫn nhất định được tuần hoàn. Mô hình truyền thẳng là dạng mạng nơ-ron đơn giản nhất vì thông tin chỉ được xử lý theo một hướng. Mặc dù dữ liệu có thể đi qua nhiều nút ẩn nhưng nó luôn di chuyển theo một hướng và không bao giờ ngược.



Hình 2.6: Mạng nơ-ron truyền thẳng.

2.1.2.2 Mạng hồi quy (Recurrent neural network)

Mạng nơ-ron tuần hoàn (RNN) là một loại mạng Nơ-ron, được sử dụng rộng rãi để thực hiện quá trình phân tích trình tự vì RNN được thiết kế để trích xuất thông tin ngữ cảnh bằng cách xác định sự phụ thuộc giữa các tem thời gian khác nhau. RNN bao gồm nhiều lớp lặp lại liên tiếp và các lớp này được lập mô hình tuần tự để ánh xạ trình tự với các trình tự khác.



Hình 2.7: Mạng nơ-ron hồi quy (Recurrent neural network)

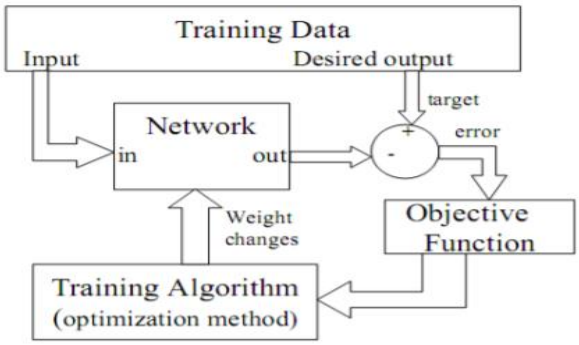
Trong các ứng dụng khác, cách chạy động tạo thành đầu ra của mạng thì những sự thay đổi các giá trị kích hoạt là đáng quan tâm.

2.1.3 Các phương pháp huấn luyện mạng nơ-ron

Một mạng nơ-ron được huấn luyện sao cho với một tập các vector đầu vào X, mạng có khả năng tạo ra tập các vector đầu ra mong muốn Y của nó. Tập X được sử dụng cho huấn luyện mạng được gọi là tập huấn luyện (training set). Các phần tử x thuộc X được gọi là các mẫu huấn luyện (training example). Quá trình huấn luyện bản chất là sự thay đổi các trọng số liên kết của mạng.

2.1.3.1 Học có giám sát (Supervised Learning)

Mạng được huấn luyện [12] bằng cách cung cấp cho nó các cặp mẫu đầu vào và các đầu ra mong muốn (target values). Các cặp được cung cấp bởi "thầy giáo", hay bởi hệ thống trên đó mạng hoạt động. Sự khác biệt giữa các đầu ra thực tế so với các đầu ra mong muốn được thuật toán sử dụng để thích ứng các trọng số trong mạng.



Hình 2.8: Mô hình học có giám sát (Supervised learning model)

2.1.3.2 Học không giám sát (Unsupervised Learning).

Là việc học không cần có bất kỳ một sự giám sát nào. Trong bài toán học không giám sát, tập dữ liệu huấn luyện được cho dưới dạng:

D = {(x1 , x2, ..., xN)}, với (x1 , x2 , ..., xN) là vector đặc trưng của mẫu huấn luyện. Nhiệm vụ của thuật toán là phải phân chia tập dữ liệu D thành các nhóm con, mỗi nhóm chứa các vector đầu vào có đặc trưng giống nhau. Như vậy với học không giám sát, số lớp phân loại chưa được biết trước, và tùy theo tiêu chuẩn đánh giá độ tương tự giữa các mẫu mà ta có thể có các lớp phân loại khác nhau.

2.1.3.3 Học củng cố (Reinforcement learning).

Đôi khi còn được gọi là học thưởng-phạt [12][13] (rewardpenalty learning), là sự tổ hợp của cả hai mô hình trên. Phương pháp này cụ thể như sau: với vector đầu vào, quan sát vector đầu ra do mạng tính được. Nếu kết quả được xem là “tốt” thì mạng sẽ được thưởng theo nghĩa tăng các trọng số kết nối lên; ngược lại mạng sẽ bị phạt, các trọng số kết nối không thích hợp sẽ được giảm xuống. Do đó học tăng cường là học theo nhà phê bình (critic), ngược với học có giám sát là học theo thầy giáo (teacher).

2.1.3.4 Học có giám sát trong mạng nơ-ron

Học có giám sát có thể được xem như việc xấp xỉ một ánh xạ: X⟶ Y, trong đó X là tập các vấn đề và Y là tập các lời giải tương ứng cho vấn đề đó. Các mẫu (x, y) với x = (x_1, x_2, . . ., x_n) ∈ X, y = (y_l, y_2, . . ., y_m) ∈ Y được cho trước. Học có giám sát trong các mạng nơ-ron thường được thực hiện theo các bước sau:

B1: Xây dựng cấu trúc thích hợp cho mạng nơ-ron, chẳng hạn có (n + 1) nơ-ron vào (n nơ-ron cho biến vào và 1 nơ-ron cho ngưỡng x_0), m nơ-ron đầu ra, và khởi tạo các trọng số liên kết của mạng.

B2: Đưa một vector x trong tập mẫu huấn luyện X vào mạng

B3: Tính vector đầu ra o của mạng

B4: So sánh vector đầu ra mong muốn y (là kết quả được cho trong tập huấn luyện) với vector đầu ra o do mạng tạo ra; nếu có thể thì đánh giá lỗi.

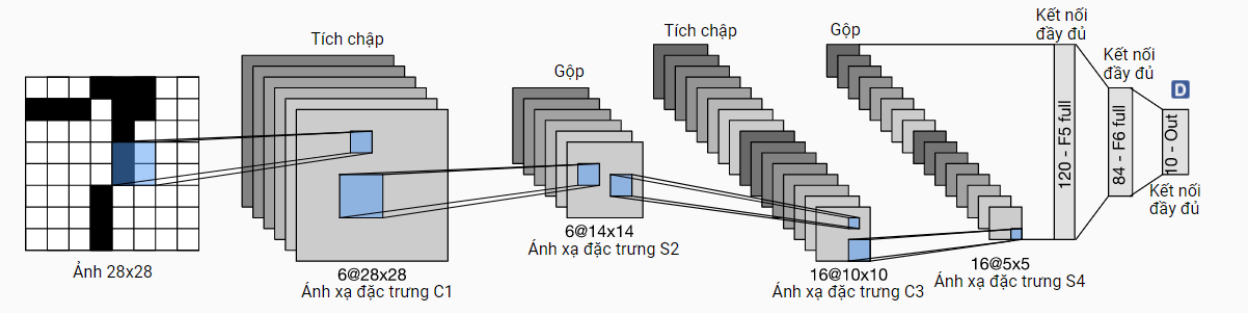
B5: Hiệu chỉnh các trọng số liên kết theo một cách nào đó sao cho ở lần tiếp theo khi đưa vector x vào mạng, vector đầu ra o sẽ giống với y hơn.

B6: Nếu cần, lặp lại các bước từ 2 đến 5 cho tới khi mạng đạt tới trạng thái hội tụ. Việc đánh giá lỗi có thể thực hiện theo nhiều cách, cách dùng nhiều nhất là sử dụng lỗi tức thời: Err = (o - y), hoặc Err = |o - y|; lỗi trung bình bình phương (MSE: mean-square error): Err = (o - y)2/2;Có hai loại lỗi trong đánh giá một mạng nơ-ron. Thứ nhất, gọi là lỗi rõ ràng (apparent error), đánh giá khả năng xấp xỉ các mẫu huấn luyện của một mạng đã được huấn luyện.

2.1.3.5 Mạng nơ-ron tích chập (CNN)

Một Mạng nơ-ron tích chập bao gồm 3 lớp chính: lớp tích chập (Convolution), lớp gộp (pooling) và lớp kết nối đầy đủ (full-connected).

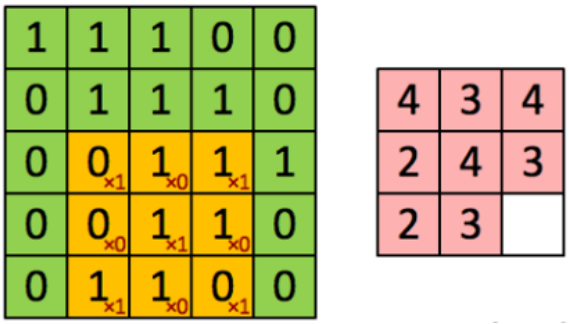
Một cách đơn giản, ta có thể xem LeNet gồm hai phần: (i) một khối các tầng tích chập; và (ii) một khối các tầng kết nối đầy đủ. Trước khi đi vào các chi tiết cụ thể, hãy quan sát tổng thể mô hình bên dưới



Hình 2.9: Mô phỏng mạng nơ-ron tích chập

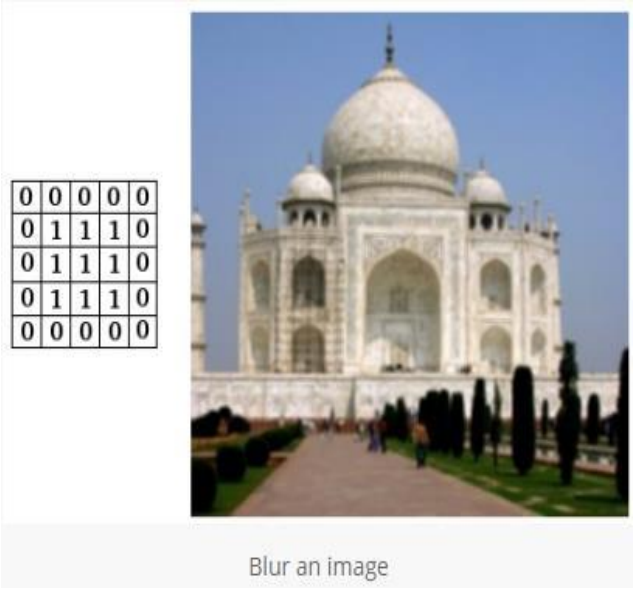
2.1.3.6 Lớp tích chập (Convolution)

Tích chập được sử dụng đầu tiên trong xử lý tín hiệu số (Signal processing). Nhờ vào nguyên lý biến đổi thông tin, các nhà khoa học đã áp dụng kĩ thuật này vào xử lý ảnh và video số. Để dễ hình dung, ta có thể xem tích chập như một cửa sổ trượt (sliding window) áp đặt lên một ma trận. Bạn có thể theo dõi cơ chế của tích chập qua hình minh họa bên dưới.



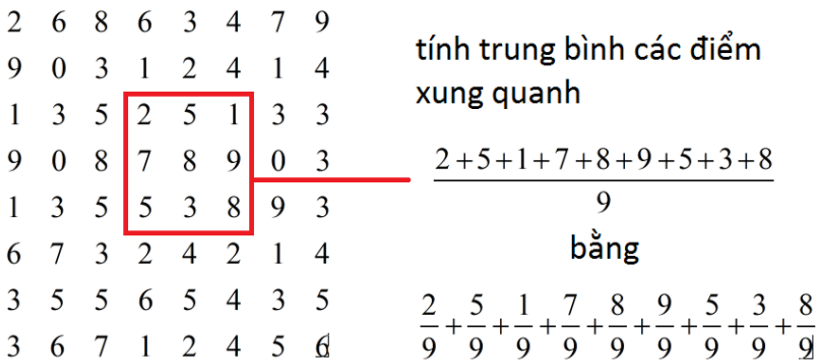
Hình 2.10: Minh họa tích chập

Kết quả của tích chập là một ma trận [13] (convoled feature) sinh ra từ việc trượt ma trận filter và thực hiện tích chập cùng lúc lên toàn bộ ma trận ảnh bên trái. Dưới đây là một vài ví dụ của phép toán tích chập.



Hình 2. 11: Ảnh mờ sau khi chập

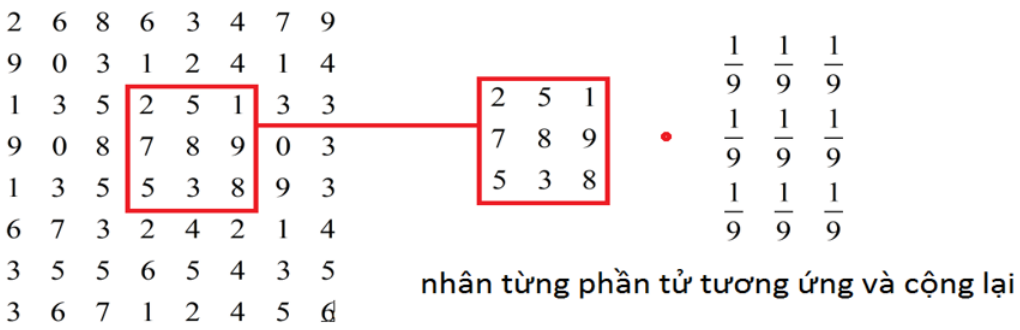
Về bản chất thực hiện làm mờ ảnh chính là tạo ra ảnh mới sao cho giá trị mức xám của mỗi pixel ở ảnh mới đúng bằng giá trị trung bình của điểm tương ứng và 8 điểm lân cận trên ảnh ban đầu.



Hình 2.12: Mô phỏng số làm mờ ảnh (bước 1)

Vậy có nghĩa là ta đang tính trung bình cộng của 9 pixel (pixel tại điểm đó và 8 pixel lân cận), vậy phép tính đó cũng giống như nhân từng giá trị mức sáng của các pixel lân cận với 1/9 sau đó cộng lại với nhau.

Áp dụng tương tự cho mọi pixel trên ảnh ban đầu và lấy từng kết quả cho từng pixel của ảnh mới, ta sẽ được ảnh mới chính là ảnh mờ của ảnh ban đầu.



Hình 2.13: Mô phỏng số làm mờ ảnh (bước 2)

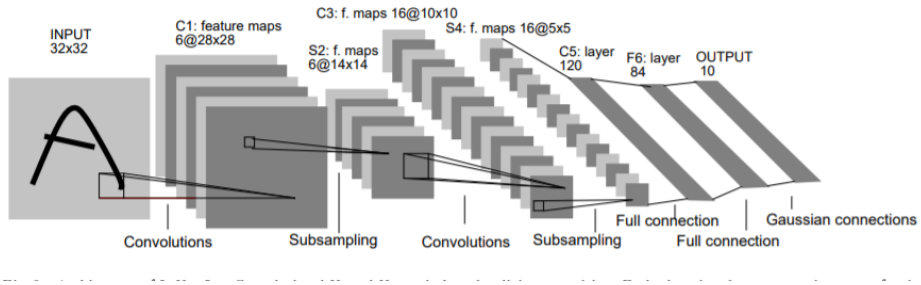
Ngoài ra, ta có thể phát hiện biên cạnh bằng cách tính vi phân (độ dị biệt) giữa các điểm ảnh lân cận.

Hình 2.7: Ảnh được phát hiện biên sau khi chập

2.2 Các công trình liên quan

2.2.1 Mạng LeNet-5 (1998)

Lenet-5 là một trong những [15] mạng CNN đầu tiên được Yann Lecun và các cộng sự phát triển vào năm 1998 để xử lý hình ảnh.

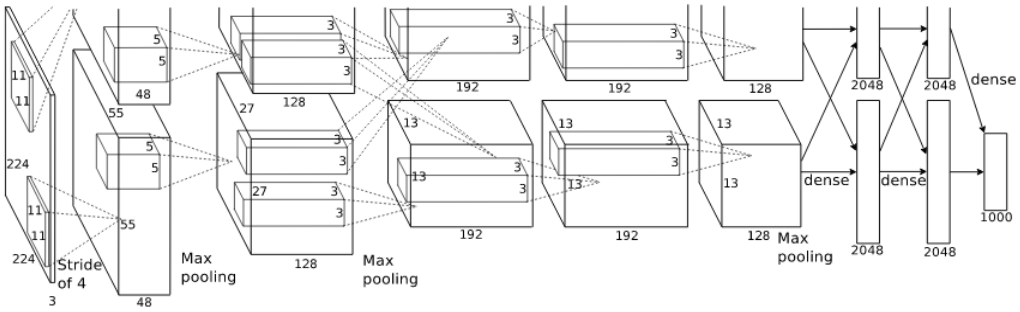


Hình 2.15: Kiến trúc LeNet-5

Lenet-5 gồm 7 lớp.

2.2.2 AlexNet

AlexNet là một mô hình mạng nơ-ron tích chập được đề xuất vào năm 2012 bởi Alex Krizhevsky.

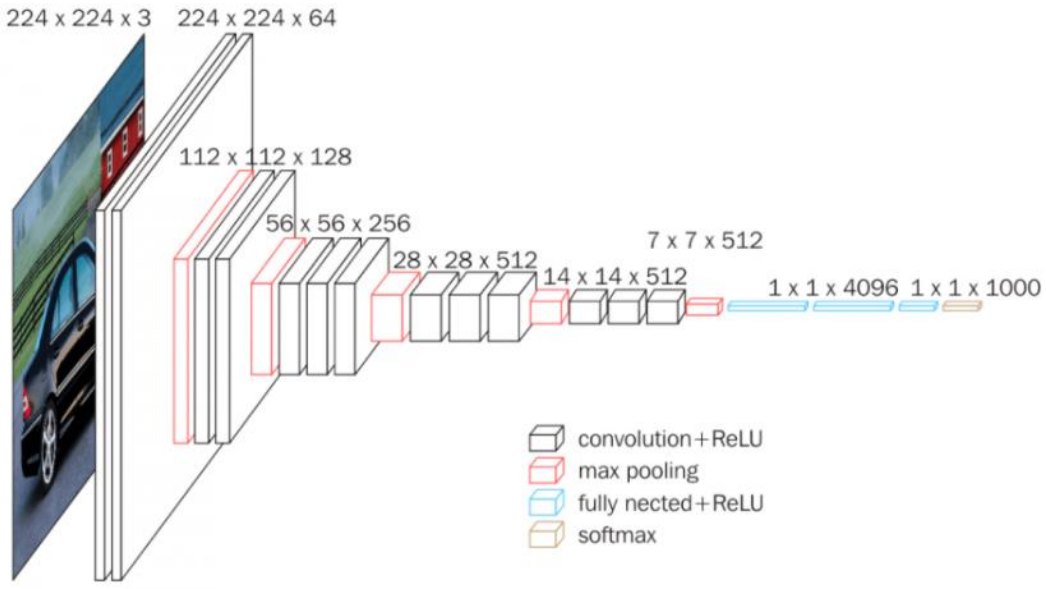


Hình 2.16: Kiến trúc AlexNet

AlexNet là một kiến trúc sâu, đầu vào cho mô hình này là hình ảnh có kích thước 227x227x3.

2.2.3 VGG-16

Khác với các mạng CNN được nghiên cứu trước đó, VGG-16 được tổ chức chặt chẽ và có số lớp gia tăng một cách đột biến thông qua việc tổ chức các ma trận lọc nhỏ hơn. Cụ thể kiến trúc của VGG-16 được mô tả trong hình 2.2 và các thông số đầu của VGG-16 được chúng tôi tính toán và phân tích chi tiết trong 16 lớp cụ thể như sau:



Hình 2.17: Kiến trúc VGG-16

2.3 Kết luận chương

Sau khi phân tích, làm rõ các khái niệm liên quan đến mạng nơ-ron tích chập và mô hình học tích cực, chúng tôi tiếp tục đề cập đến 3 mạng nơ-ron tích chập phổ biến. Mỗi mạng nơ-ron có ưu và nhược điểm riêng.

- Thứ nhất là mạng LeNet-5. Mạng này khá đơn giản, LeNet-5 có 7 lớp. trong đó có ba lớp chập (C1, C3 và C5) và hai lớp gộp (S2 và S4), và 1 lớp được kết nối đầy đủ (F6), tiếp theo là lớp đầu ra. LeNet-5 sử dụng hàm sigmoid (tanh) ở mỗi lớp tích chập. Chính vì thế, tốc độ tính toán của LeNet-5 chậm.

- Thứ hai là mạng AlexNet. AlexNet bao gồm 8 lớp (5 lớp tích chập và 3 lớp kết nối đầy đủ). AlexNet l có một số đặc điểm của AlexNet cho phép giảm thời gian đào tạo 6 lần (so với tanh) khi so sánh cùng độ chính xác. AlexNet có cấu trúc tương tự như LeNet, nhưng vì sử dụng nhiều tầng tích chập hơn, do vậy AlexNet có khả năng xử lý bộ dữ liệu với tham số truyền vào lớn hơn LeNet.

- Thứ ba là mạng VGG-16. VGG-16 có cửa sổ tích chập nhỏ (3×3). Chính vì sử dụng cửa sổ tích chập nhỏ nên rõ ràng VGG-16 cho hiệu quả cao hơn mạng có cửa sổ tích chập rộng nhưng ít lớp.

CHƯƠNG 3: MÔ HÌNH NHẬN BIẾT CHỮ KÝ VIẾT TAY

3.1 Mô phỏng chương trình nhận biết chữ ký viết tay

3.1.1 Giới thiệu chung

- Chương trình nhận dạng chữ ký viết tay dựa trên phương thức học có giám sát (Supervised Learning) theo dạng phân loại (Classification).

3.1.2 Công cụ sử dụng

- Công cụ: Jupyter Notebook. Trước đây là nó có tên Ipython Notebook, đến năm 2014 đổi lại thành Jupyter Notebook. Jupyter hỗ trợ rất nhiều các kernel cho các ngôn ngữ khác nhau, khoảng trên 40 ngôn ngữ trong đó có Python.
- Cài đặt công cụ Jupyter Notebook: Cài đặt Python => Cài đặt Jupyter Notebook bằng lệnh: pip install jupyter.
- Tập dữ liệu: tập chữ ký của người việt thu thập được 160 mẫu tương ứng với 160 người. Tỷ lệ trong 1 mẫu là 24 chữ ký thật và 30 chữ ký giả.

3.2 Môi trường mô phỏng thực nghiệm

Môi trường:

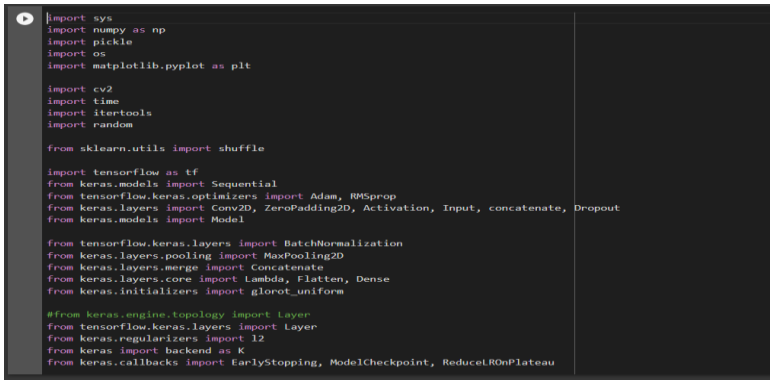
- Chương trình được thực nghiệm trên Windows 10, máy tính laptop core i7 tốc độ 3.4GHz , bộ nhớ RAM 8GB. Sử dụng bộ thư viện sklearn, tensoflow, keras, cv2 phiên bản 2.7 và các thư viện hỗ trợ khác như: os, time, numpy, matplotlib. Đây đều là các thư viện mã nguồn mở, sử dụng ổn định trên Python 3.8.

Mô phỏng thực nghiệm:

- Mô tả: Đưa tập dữ liệu chữ ký vào mô hình mạng CNN để lấy ra một cặp chữ ký và dựa vào ngưỡng trên khoảng cách để xác định hình ảnh sau là chữ ký thật hay chữ ký giả so với ảnh gốc.

Thực nghiệm và kết quả mô phỏng:

- **Bước 1:** Import các thư viện cần thiết



```
import sys
import numpy as np
import pickle
import os
import matplotlib.pyplot as plt

import cv2
import time
import itertools
import random

from sklearn.utils import shuffle

import tensorflow as tf
from keras.models import Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from keras.layers import Conv2D, ZeroPadding2D, Activation, Input, concatenate, Dropout
from keras.models import Model

from tensorflow.keras.layers import BatchNormalization
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import Concatenate
from keras.layers.core import Lambda, Flatten, Dense
from keras.initializers import glorot_uniform

#from keras.engine.topology import Layer
from tensorflow.keras.layers import Layer
from keras.regularizers import l2
from keras import backend as K
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

Hình 3.1: Quá trình import thư viện

- Thư viện tensorflow, keras, sklearn đều là phiên bản 2.7.

Bước 2: Đưa các hình ảnh từ tập dữ liệu thu thập vào thành 2 nhóm:

```
#path = "./BHSig260/Hindi/"
path = "./BHSig260/Vietnam/"
```

```
# Lấy ra danh sách các thư mục và sắp xếp
dir_list = next(os.walk(path))[1]
dir_list.sort()
```

```
# Mỗi người tách biệt chữ ký thật và chữ ký giả
# Chữ ký thật đc lưu vào danh sách "orig_groups"
# Chữ ký giả đc lưu vào danh sách "forged_groups"
orig_groups, forg_groups = [], []
for directory in dir_list:
    images = os.listdir(path+directory)
    images.sort()
    images = [path+directory+'/'+x for x in images]
    forg_groups.append(images[:30]) # First 30 signatures in each folder are forrged
    orig_groups.append(images[30:]) # Next 24 signatures are genuine
```

Hình 3.2 Chia dữ liệu đầu vào thành hai nhóm thật và giả

```
# All the images will be converted to the same size before processing
img_h, img_w = 155, 220
```

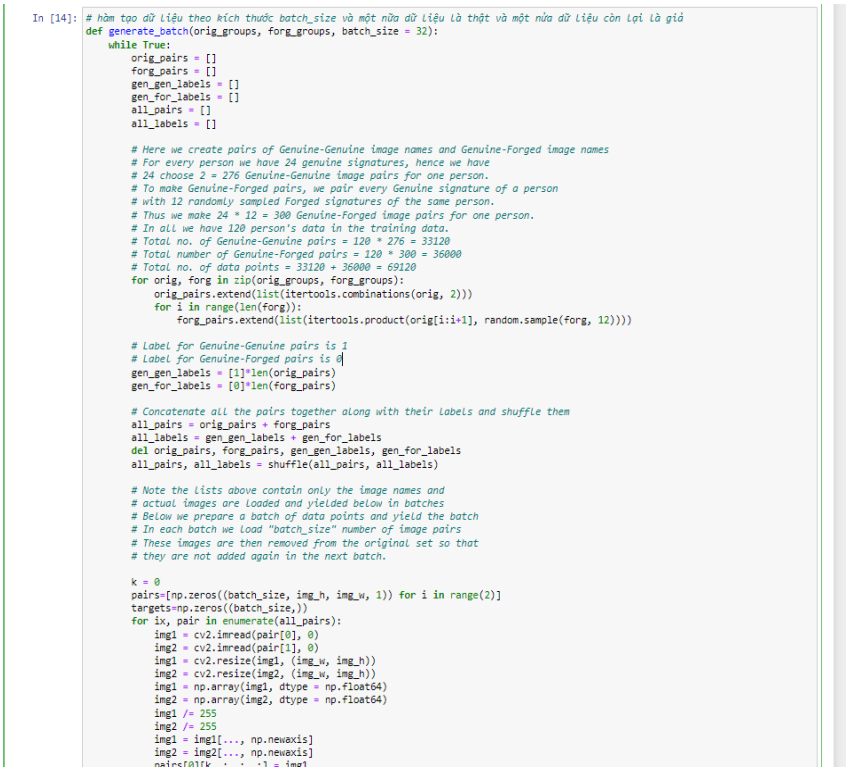
```
# hàm có chức năng chọn ngẫu nhiên một chữ ký từ tập train và thiết lập ra 2 bản sao thật và giả
def visualize_sample_signature():
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (10, 10))
    k = np.random.randint(len(orig_train))
    orig_img_names = random.sample(orig_train[k], 2)
    forg_img_name = random.sample(forg_train[k], 1)
    orig_img1 = cv2.imread(orig_img_names[0], 0)
    orig_img2 = cv2.imread(orig_img_names[1], 0)
    forg_img = plt.imread(forg_img_name[0], 0)
    orig_img1 = cv2.resize(orig_img1, (img_w, img_h))
    orig_img2 = cv2.resize(orig_img2, (img_w, img_h))
    forg_img = cv2.resize(forg_img, (img_w, img_h))

    ax1.imshow(orig_img1, cmap = 'gray')
    ax2.imshow(orig_img2, cmap = 'gray')
    ax3.imshow(forg_img, cmap = 'gray')

    ax1.set_title('Genuine Copy')
    ax1.axis('off')
    ax2.set_title('Genuine Copy')
    ax2.axis('off')
    ax3.set_title('Forged Copy')
    ax3.axis('off')
```

Hình 3.3: Hàm lấy ra hai chữ thật và một chữ ký giả từ bộ dữ liệu đã được chia

- **Bước 3:** Với mỗi một bộ có 24 chữ ký thật thì chọn ra 2 chữ ký thì có 276 cặp chữ ký thật - chữ ký thật. Đối với cặp chữ ký thật - chữ ký giả thì lấy cặp chữ ký thật của 1 người với 12 chữ ký giả của người đó nên có 300 bộ.



```
In [14]: # hàm tạo dữ liệu theo kích thước batch_size và một nửa dữ liệu là thật và một nửa dữ liệu còn lại là giả
def generate_batch(orig_groups, forg_groups, batch_size = 32):
    while True:
        orig_pairs = []
        forg_pairs = []
        gen_gen_labels = []
        gen_for_labels = []
        all_pairs = []
        all_labels = []

        # Here we create pairs of Genuine-Genuine image names and Genuine-Forged image names
        # For every person we have 24 genuine signatures, hence we have
        # 24 choose 2 = 276 Genuine-Genuine image pairs for one person.
        # To make Genuine-Forged pairs, we pair every Genuine signature of a person
        # with 12 randomly sampled Forged signatures of the same person.
        # Thus we make 24 * 12 = 300 Genuine-Forged image pairs for one person.
        # In all we have 120 person's data in the training data.
        # Total no. of Genuine-Genuine pairs = 120 * 276 = 33120
        # Total number of Genuine-Forged pairs = 120 * 300 = 36000
        # Total no. of data points = 33120 + 36000 = 69120
        for orig, forg in zip(orig_groups, forg_groups):
            orig_pairs.extend(list(itertools.combinations(orig, 2)))
            for i in range(len(forg)):
                forg_pairs.extend(list(itertools.product(orig[i:i+1], random.sample(forg, 12))))

        # Label for Genuine-Genuine pairs is 1
        # Label for Genuine-Forged pairs is 0
        gen_gen_labels = [1]*len(orig_pairs)
        gen_for_labels = [0]*len(forg_pairs)

        # Concatenate all the pairs together along with their labels and shuffle them
        all_pairs = orig_pairs + forg_pairs
        all_labels = gen_gen_labels + gen_for_labels
        del orig_pairs, forg_pairs, gen_gen_labels, gen_for_labels
        all_pairs, all_labels = shuffle(all_pairs, all_labels)

        # Note the lists above contain only the image names and
        # actual images are loaded and yielded below in batches
        # Below we prepare a batch of data points and yield the batch
        # In each batch we load "batch_size" number of image pairs
        # These images are then removed from the original set so that
        # they are not added again in the next batch.

        k = 0
        pairs=[np.zeros((batch_size, img_h, img_w, 1)) for i in range(2)]
        targets=np.zeros((batch_size,))
        for ix, pair in enumerate(all_pairs):
            img1 = cv2.imread(pair[0], 0)
            img2 = cv2.imread(pair[1], 0)
            img1 = cv2.resize(img1, (img_w, img_h))
            img2 = cv2.resize(img2, (img_w, img_h))
            img1 = np.array(img1, dtype = np.float64)
            img2 = np.array(img2, dtype = np.float64)
            img1 /= 255
            img2 /= 255
            img1 = img1[..., np.newaxis]
            img2 = img2[..., np.newaxis]
```

Hình 3.4: Hàm tạo các cặp dữ liệu theo kích thước batch_size.

- **Bước 4:** Mô hình CNN

- Sequential(): khởi tạo model.
- Lớp Convolution2D với 96 bộ lọc, kích thước bộ lọc 11x11, activation=’relu’ lọc ra các giá trị nhỏ hơn 0. Tốc độ hội tụ và tính toán nhanh hơn các hàm khác.
- Lớp Dropout dùng để bỏ qua 1 vài unit trong quá trình trainning và mục đích chính là để chống over-fitting.
- Lớp Flatten là lớp reshape trong đó tất cả các axes được làm phẳng hoặc được ghép lại với nhau.
- Lớp Dense thể hiện việc tất cả các unit của layer trước được nối toàn bộ với các unit của hiện tại

```
# Mô hình CNN
def create_base_network_signet(input_shape):
    seq = Sequential()
    seq.add(Conv2D(96, kernel_size=(11, 11), activation='relu', name='conv1_1', strides=4, input_shape= input_shape, kernel_initi
    seq.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(ZeroPadding2D((2, 2), data_format="channels_last"))

    seq.add(Conv2D(128, kernel_size=(5, 5), activation='relu', name='conv2_1', strides=1, kernel_initializer='glorot_uniform'))
    seq.add(BatchNormalization(epsilon=1e-06, axis=1, momentum=0.9))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(Dropout(0.3))# added extra
    seq.add(ZeroPadding2D((1, 1), data_format="channels_last"))

    seq.add(Conv2D(256, kernel_size=(3, 3), activation='relu', name='conv3_1', strides=1, kernel_initializer='glorot_uniform'))
    seq.add(ZeroPadding2D((1, 1), data_format="channels_last"))

    seq.add(Conv2D(128, kernel_size=(3, 3), activation='relu', name='conv3_2', strides=1, kernel_initializer='glorot_uniform'))
    seq.add(MaxPooling2D((3,3), strides=(2, 2)))
    seq.add(Dropout(0.3))# added extra
    seq.add(Flatten(name='flatten'))
    seq.add(Dense(512, activation='relu', kernel_initializer='glorot_uniform'))
    seq.add(Dropout(0.5))
    seq.add(Dense(1))

    return seq
```

Hình 3.5: Mô hình CNN

- Bước 5: Tranning model

  Chọn Epoch = 10 là số lần chạy để đưa độ chính xác cao nhất.

```
#trainning dữ liệu
# compile model using RMSProp Optimizer and Contrastive loss function defined above
rms = RMSprop(lr=1e-4, rho=0.9, epsilon=1e-08)
#model.compile(loss=contrastive_loss, optimizer=rms)
model.compile(optimizer=rms, loss='categorical_crossentropy', metrics=['accuracy'])
#model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
c:\users\hoangnam\appdata\local\programs\python\python38\lib\site-packages\keras\optimizer_v2\rmsprop.py:130: UserWarning: The
`lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
```

```
results = model.fit(generate_batch(orig_train, forg_train, batch_sz),
                            steps_per_epoch = num_train_samples//batch_sz,
                            epochs = 10,
                            validation_data = generate_batch(orig_val, forg_val, batch_sz),
                            validation_steps = num_val_samples//batch_sz
                            )
```

Hình 3.6: Tranning model

- **Bước 6:** Xây dựng hàm test dựa vào ngưỡng trên khoảng cách

Dựa vào ngưỡng trên khoảng cách, hình ảnh lấy được so sánh với hình ảnh gốc. Nếu ngưỡng xác định được lớn hơn ngưỡng sau khi tranning model thì đó là chữ ký giả và ngược lại là chữ ký thật

```python
# tính toán độ chính xác và ngưỡng trên khoảng cách
def compute_accuracy_roc(predictions, labels):
    dmax = np.max(predictions)
    dmin = np.min(predictions)
    nsame = np.sum(labels == 1)
    ndiff = np.sum(labels == 0)

    step = 0.01
    max_acc = 0
    best_thresh = -1

    for d in np.arange(dmin, dmax+step, step):
        idx1 = predictions.ravel() <= d
        idx2 = predictions.ravel() > d

        tpr = float(np.sum(labels[idx1] == 1)) / nsame
        tnr = float(np.sum(labels[idx2] == 0)) / ndiff
        acc = 0.5 * (tpr + tnr)
#       print ('ROC', acc, tpr, tnr)

        if (acc > max_acc):
            max_acc, best_thresh = acc, d

    return max_acc, best_thresh
```

```python
test_gen = generate_batch(orig_test, forg_test, 1)
pred, tr_y = [], []
for i in range(num_test_samples):
    (img1, img2), label = next(test_gen)
    tr_y.append(label)
    pred.append(model.predict([img1, img2])[0][0])
```

```python
# đưa ra độ chính xác và ngưỡng xác định thật giả dựa vào model sau khi trainnin
tr_acc, threshold = compute_accuracy_roc(np.array(pred), np.array(tr_y))
tr_acc, threshold
#Nếu lớn hơn threshold thì là chữ ký giả
```

Hình 3.7: Tính toán ngưỡng và độ chính xác sau khi train model

Kết quả: độ chính xác lên đến 100% và ngưỡng là 0.0 do tập dữ liệu thu thập được ở phần chữ ký thật là 24 hình ảnh giống nhau

- **Bước 7:** Xây dựng hàm kiểm tra chữ ký thật giả dựa vào ngưỡng sau khi train model ở bước 6

```python
# hàm demo kiểm tra chữ ký thật giả
def predict_score():
    test_point, test_label = next(test_gen)
    img1, img2 = test_point[0], test_point[1]

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 10))
    ax1.imshow(np.squeeze(img1), cmap='gray')
    ax2.imshow(np.squeeze(img2), cmap='gray')
    ax1.set_title('Genuine')
    if test_label == 1:
        ax2.set_title('Test')
        #ax2.set_title('Forged')
    else:
        ax2.set_title('Test')
        #ax2.set_title('Genuine')
    ax1.axis('off')
    ax2.axis('off')
    plt.show()
    result = model.predict([img1, img2])
    diff = result[0][0]
    print("Difference Score = ", diff)
    if diff > threshold:
        print("Its a Forged Signature")
        #print("Its a Genuine Signature")
    else:
        print("Its a Genuine Signature")
        #print("Its a Forged Signature")
```

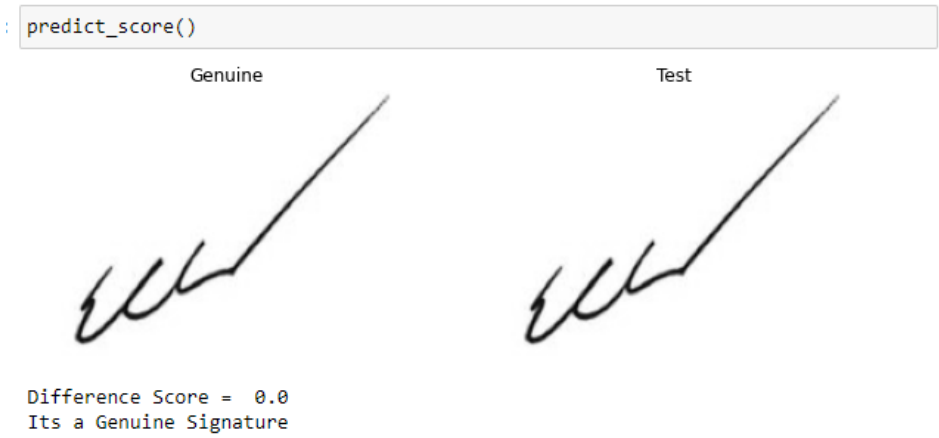
Hình 3.8: Hàm kiểm tra chữ ký thật giả

Kết quả mô phỏng:

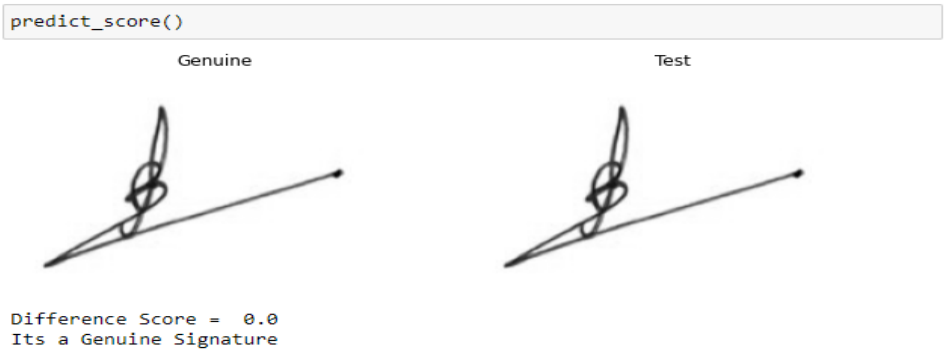
- Chữ ký thật

Hình 3.9: Kết quả chữ ký thật lần 1

Hình 3.10: Kết quả chữ ký thật lần 2



Hình 3.11: Kết quả chữ ký thật lần 3



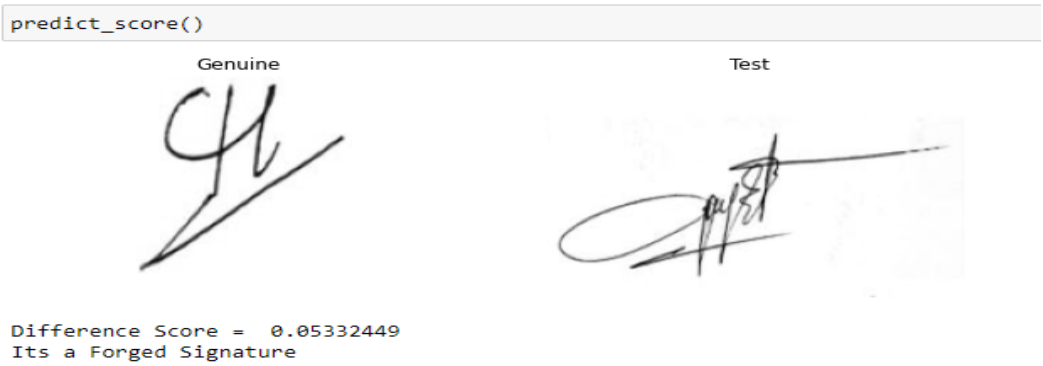
Hình 3.12: Kết quả chữ ký thật lần 4

Nhận xét: do đầu vào chữ ký thật là 24 hình ảnh giống nhau lên ngưỡng trên khoảng cách luôn là 0.0. Ngưỡng trên khoảng cách luôn >= 0.0.

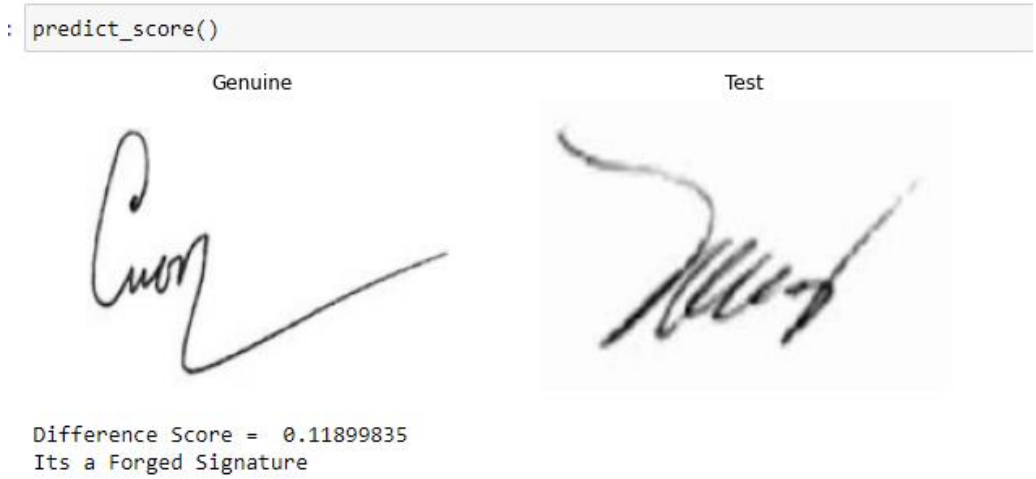
- Chữ ký giả

Hình 3.14: Kết quả chữ ký giả lần 1

Hình 3.15 Kết quả chữ ký giả lần 2



Hình 3.16: Kết quả chữ ký giả lần 3



Hình 3.17: Kết quả chữ ký giả lần 4

Kết luận:

Mô hình CNN được huấn luyện với tốc độ học lr = 1e-4, sử dụng các hàm kích hoạt relu ở các lớp tích chập và điều chỉnh được các tham số trong các lớp mạng để có được độ chính xác cao nhất.

3.3 Kết luận chương.

Sử dụng mô hình CNN để training model giúp cải thiện độ chính xác hơn, cải thiện độ tin cậy hơn so với các mô hình machine learning trước đây.

Phương pháp	Độ chính xác
Nearest-neighbor	75%
LIBSVM	77%
1-layer Neural nets	79%
CNN	80%

Từ bảng so sánh [16] trên ta nhận thấy mô hình CNN đưa ra kết quả với độ chính xác cao nhất.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Qua luận văn này, Chúng tôi không những phân tích mô hình, thuật toán, và dữ liệu mà còn khảo sát, đánh giá các lớp, khối hàm, và các kỹ thuật trong mạng nơ-ron tích chập hiện đại. Việc này đáp ứng được cho sự thay đổi kiến trúc, thuật toán có thể giải quyết các bài toán đơn giản, phù hợp với các ứng dụng phân loại chữ ký việt nam viết tay trong thực tế.

Nội dung của luận văn thực hiện bao gồm việc cài đặt thuật toán, đánh giá mô hình và phân tích các tập dữ liệu huấn luyện và thử nghiệm. Thông qua một số kịch bản mô phỏng và đánh giá hiệu năng thực thi của hệ thống, các gợi mở cho việc tối ưu một hệ thống thực dựa trên tập mã nguồn mở đã sẵn sàng triển khai cho một hệ thống phân loại các chữ ký tiếng anh hoặc tiếng việt trong thực tế trên một nền tảng học sâu.

Trong luận văn này, các kết quả cũng đưa ra được độ nhận diện chữ ký bằng CNN có độ chính xác cao.

Hướng nghiên cứu trong tương lai của chúng tôi là hoàn thiện mô hình lý thuyết nghiên cứu, hiệu chỉnh các thuật toán, các lớp, hàm, hệ số lọc bỏ phù hợp để ứng dụng có độ chính xác cao hơn khi khảo sát hình ảnh các chữ ký viết tay thực tế ở Việt Nam. Nâng cao hiệu quả chương trình, nhận diện nhiều ký tự cùng lúc. Phát triển chương trình thành module phần cứng. Thêm vào đó, chúng tôi sẽ xem xét phát triển tập dữ liệu phù hợp với khả năng triển khai cho các ứng dụng nhận diện chữ ký viết tay cho nhu cầu, xu hướng hiện đại hóa, công nghệ 4.0 ở Việt Nam.

Chúng tôi cũng bị hạn chế ở đây bởi thực tế là bộ dữ liệu của chúng tôi tương đối nhỏ, rất khó để tìm thấy các bộ dữ liệu chữ ký có sẵn tốt. Do hạn chế về thời gian và năng lực tính toán của máy tính, luận văn không thể khảo sát hết các mô hình CNN. Vấn đề nghiên cứu trong luận văn là khá mới nên khi trình bày, dịch thuật tài liệu chắc chắn không tránh khỏi những thiếu sót và hạn chế cần phải khắc phục. Rất mong nhận được sự góp ý của độc giả.