

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN
TRÍ TUỆ NHÂN TẠO

*Đề tài: Tìm hiểu mạng nơ-ron và xây dựng chương trình
nhận dạng chó và mèo.*

GIÁO VIÊN HƯỚNG DẪN: TS. Phạm Minh Tuấn
SINH VIÊN THỰC HIỆN: Trương Phước Minh Quân

Lớp: 15TCLC2

MÃ SỐ SINH VIÊN: 102150292

Đà Nẵng, 5/2018

Mục Lục

I.	TỔNG QUAN VỀ MẠNG NƠON	4
1.	Khái niệm	4
2.	Mạng neural nhân tạo.....	7
3.	Học có giám sát và học không có giám sát	7
II.	CÁC CƠ SỞ LÝ THUYẾT	8
1.	Giải thuật lan truyền ngược	8
2.	Learning Rate	9
3.	Hàm Active	9
4.	Một số gợi ý thiết kế mạng Neural.....	11
III.	PHÂN TÍCH VÀ THIẾT KẾ DỮ LIỆU VÀ MẠNG NEURAL	12
1.	Ý tưởng.....	12
2.	Phân tích	13
3.	Thiết kế.....	14
IV.	TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ	Error! Bookmark not defined.5
1.	Khởi tạo các class	15
2.	Chương trình hoàn chỉnh	17
3.	Kết quả	25
V.	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	26
1.	Kết luận.....	26
2.	Hướng phát triển.....	27

Danh sách hình vẽ

Hình 1. Neural sinh học.....	4
Hình 2. Neural nhân tạo.....	6
Hình 3. Sơ đồ đơn giản về một mạng neural nhân tạo.....	7
Hình 4. Hàm đồng nhất.....	10
Hình 5. Hàm bước nhị phân.....	10
Hình 6. Hàm Sigmoid.....	11
Hình 7. Hàm $y = \text{Log}(1+ x)$	11
Hình 8. Hàm $y = \text{Tan}(x)$	11
Hình 9. Kaggle.....	13
Hình 10. Tập dữ liệu.....	14
Hình 11. Dữ liệu file.text.....	15
Hình 12. Thuật toán xử lý ảnh.....	16
Hình 13. Class Neural.....	16
Hình 14. Class HiddenNeural.....	17
Hình 15. Class OutputNeural.....	17
Hình 16. Class NeuralNetwork.....	17
Hình 17. Sigmoid Function.....	18
Hình 18. Kết quả train.....	26
Hình 19. Kết quả test.....	26

LỜI MỞ ĐẦU

Từ lâu các nhà khoa học đã nhận thấy những ưu điểm của bộ óc con người và tìm cách bắt chước để thực hiện trên những máy tính, tạo cho nó có khả năng học tập, nhận dạng và phân loại. Vì vậy các nhà khoa học đã nghiên cứu và sáng tạo ra mạng Neural nhân tạo. Nó thực sự được chú ý và nhanh chóng trở thành một hướng nghiên cứu đầy triển vọng đặc biệt là lĩnh vực nhận dạng. Và bài toán nhận dạng chó mèo là một bài toán con trong lớp các bài toán nhận dạng, xử lý ảnh.

Hiện nay trên thế giới, các sản phẩm nhận dạng hình ảnh đã được triển khai tương đối rộng rãi. Ở nước ta trong một vài năm gần đây cũng đã có một số sản phẩm nhận dạng hình ảnh được triển khai trên thị trường. Nhưng các sản phẩm này được bán trên thị trường dưới dạng đóng kín nên việc để phát triển thành phần mềm tự động cập nhật ảnh là điều không thể. Vì vậy nên tôi đã chọn đề tài “Tìm hiểu mạng nơ-ron và xây dựng chương trình nhận dạng chó và mèo”.

Em xin chân thành cảm ơn thầy giáo TS. Phạm Minh Tuấn đã giúp đỡ em rất nhiều trong quá trình tìm hiểu, thiết kế và hoàn thành đề tài đồ án 2 này.

Đà Nẵng, tháng 5 năm 2018

Sinh viên

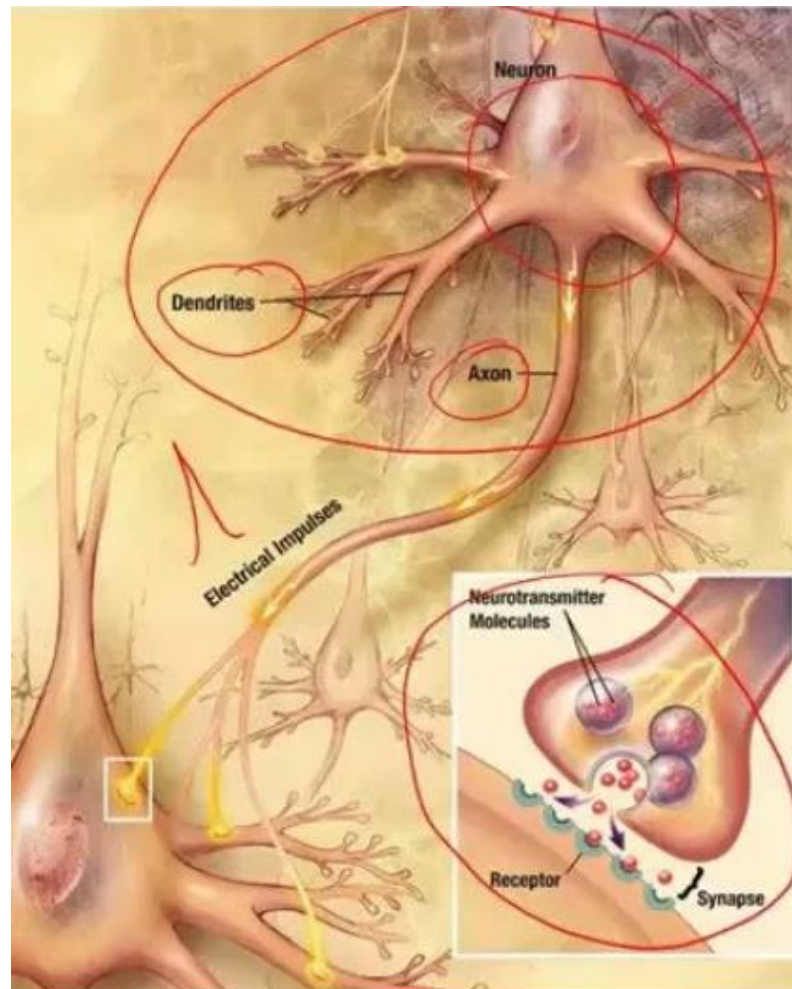
Trương Phước Minh Quân

I. TỔNG QUAN VỀ MẠNG NƠRON

1. Khái niệm

- **Neural sinh học**

Một neuron được cấu tạo gồm những thành phần chính sau: Dendrite, Soma, Synapse, Axon như hình 1.1.



Hình 1. Neural sinh học

Các dendrites là các dây mảnh, dài, gắn liền với soma, chúng truyền dữ liệu (dưới dạng xung điện thế) đến cho soma xử lý. Bên trong soma các dữ liệu đó được tổng hợp lại, có thể xem gần đúng sự tổng hợp ấy như là một phép lấy tổng tất cả các dữ liệu mà neuron nhận được.

Một loại dây dẫn tín hiệu khác cũng gắn với soma là các axon. Khác với dendrites, axons có khả năng phát các xung điện thế, chúng là các dây dẫn tín hiệu từ neuron đi các nơi khác. Chỉ khi nào điện thế

trong soma vượt quá một giá trị ngưỡng nào đó thì axon mới phát một xung điện thế, còn nếu không thì nó ở trạng thái nghỉ.

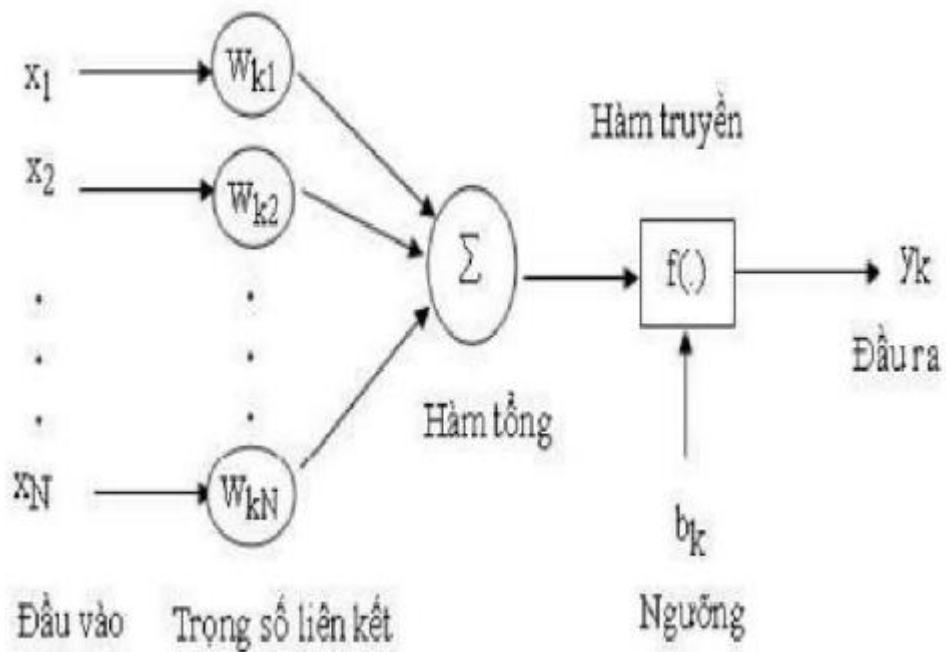
Axon nối với các dendrites của các neural khác thông qua những mối nối đặc biệt gọi là synapse. Khi điện thế của synapse tăng lên do các xung phát ra từ axon thì synapse sẽ nhả ra một số chất hoá học (neurotransmitters); các chất này mở "cửa" trên dendrites để cho các ions truyền qua. Chính dòng ions này làm thay đổi điện thế trên dendrites, tạo ra các xung dữ liệu lan truyền tới các neural khác.

Có thể tóm tắt hoạt động của một neural như sau: neural lấy tổng tất cả các điện thế vào mà nó nhận được, và phát ra một xung điện thế nếu tổng ấy lớn hơn một ngưỡng nào đó. Các neural nối với nhau ở các synapses. Synapse được gọi là mạch khi nó cho phép truyền dẫn dễ dàng tín hiệu qua các neural khác. Ngược lại, một synapse yếu sẽ truyền dẫn tín hiệu rất khó khăn.

Các synapses đóng vai trò rất quan trọng trong sự học tập. Khi chúng ta học tập thì hoạt động của các synapses được tăng cường, tạo nên nhiều liên kết mạnh giữa các neural. Có thể nói rằng người nào học càng giỏi thì càng có nhiều synapses và các synapses ấy càng mạnh mẽ, hay nói cách khác, thì liên kết giữa các neural càng nhiều, càng nhạy bén..

- **Neural nhân tạo**

Neural nhân tạo là một đơn vị tính toán có nhiều đầu vào và một đầu ra, mỗi đầu vào đến từ một liên kết. Đặc trưng của neural là một hàm kích hoạt phi tuyến chuyển đổi tổ hợp tuyến tính của tất cả các tín hiệu đầu vào thành tín hiệu đầu ra. Hàm kích hoạt này đảm bảo tính chất phi tuyến cho tính toán của mạng neural.



Hình 2. Neural nhân tạo

Một neural được cấu tạo gồm các thành phần chính : liên kết neural, bộ cộng , hàm kích hoạt.

Liên kết neural là một thành phần của mạng neural nhân tạo để liên kết giữa các neural, nó nối đầu ra của neural lớp này với đầu vào của một neural trong lớp khác. Đặc trưng của thành phần liên kết là một trọng số mà mỗi tín hiệu đi qua đều được nhân với trọng số này. Các trọng số liên kết chính là các tham số tự do cơ bản của mạng neuron, có thể thay đổi được nhằm thích nghi với môi trường xung quanh.

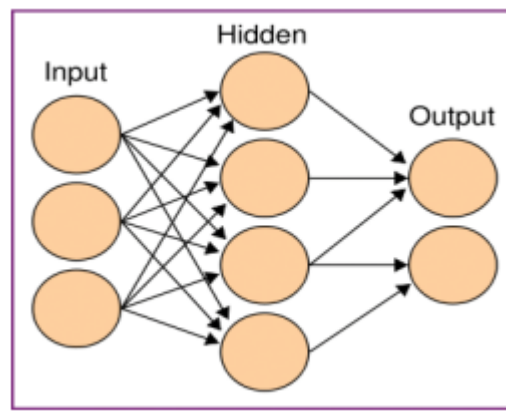
Bộ cộng dùng để tính tổng các tín hiệu đầu vào của neural, đã được nhân với các trọng số liên kết tương ứng. phép toán được mô tả ở đây tạo nên một bộ hợp tuyến tính. Hàm kích hoạt hay còn gọi hàm kích hoạt phi tuyến, chuyển đổi một tổ hợp tuyến tính của tất cả các tín hiệu đầu vào thành tín hiệu đầu ra.

Hàm kích hoạt này đảm bảo tính chất phi tuyến cho tính toán mạng neural. Nó được xem như là một hàm giới hạn, nó giới hạn phạm vi biên độ cho phép của tín hiệu đầu ra trong một khoảng giá trị hữu hạn. Mô hình neural trong hình 2 còn bao gồm một hệ số điều chỉnh b tác động từ bên ngoài. Hệ số điều chỉnh b có tác dụng tăng lên hoặc giảm đi đầu vào thực của hàm kích hoạt, tùy theo nó dương hay âm

2. Mạng Neural nhân tạo

Là một hệ thống bao gồm nhiều phần tử xử lý đơn giản (hay còn gọi là neural) tựa như neural thần kinh của não người, hoạt động song song và được nối với nhau bởi các liên kết neural. Mỗi liên kết kèm theo một trọng số nào đó, đặc trưng cho tính kích hoạt hoặc ức chế giữa các neural.

Có thể xem các trọng số là phương tiện để lưu trữ thông tin dài hạn trong mạng neural và nhiệm vụ của quá trình huấn luyện của mạng là cập nhật các trọng số khi có thêm thông tin về mẫu học. Hay nói một cách khác, các trọng số đều được điều chỉnh sao cho đáng điều vào ra của mạng sẽ mô phỏng hoàn toàn phù hợp với môi trường đang xem xét.



Hình 3. Sơ đồ đơn giản về một mạng neural nhân tạo

Mô hình mạng neural ở trên gồm 3 lớp: lớp nhập (input), lớp ẩn (hidden) và lớp xuất (output). Mỗi nút trong lớp nhập nhận giá trị của một biến độc lập và chuyển vào mạng.

Dữ liệu từ tất cả các nút trong lớp nhập được tích hợp – ta gọi là tổng trọng số và chuyển kết quả cho các nút trong lớp ẩn. Gọi là “ẩn” vì các nút trong lớp này chỉ liên lạc với các nút trong lớp nhập và lớp xuất, và chỉ có người thiết kế mạng mới biết lớp này (người sử dụng không biết lớp này).

Các nút trong lớp xuất nhận các tín hiệu tổng trọng số từ các nút trong lớp ẩn. Mỗi nút trong lớp xuất tương ứng với một biến phụ thuộc.

3. Học có giám sát và học không có giám sát

Một mạng neural cần được đào tạo trước khi nó có thể được đưa vào sử dụng. Huấn luyện liên quan đến việc đưa vào mạng nơron mẫu huấn luyện và cho phép nó tìm hiểu bằng cách hiệu chỉnh trọng số liên kết và các thông số

khác nhau. Mạng neural có thể được phân loại thành 2 loại dựa vào loại học tập.

- **Mạng neural học có giám sát**

Trong phương pháp học có giám sát, mạng neural học từ các mẫu. Tập huấn luyện bao gồm 1 tập hợp các mẫu đầu vào và các kết quả đầu ra mong muốn tương ứng với mẫu đầu vào. Các mạng neural điều chỉnh trọng số liên kết của nó để tìm hiểu mối quan hệ giữa các cặp đầu vào – đầu ra. Mạng neural được huấn luyện thành công thì có thể được sử dụng để tìm đầu ra phù hợp nhất đối với bất kỳ đầu vào hợp lệ.

- **Mạng neural học không có giám sát**

Mạng neural chỉ nhận được một tập hợp các đầu vào từ môi trường bên ngoài. Nó có vẻ bí ẩn để tưởng tượng những gì các mạng có thể học hỏi từ thiết lập một chỉ số đầu vào. Tuy nhiên, có thể chứng minh rằng một mạng lưới không có giám sát có thể xây dựng đại diện của các đầu vào có thể được sử dụng cho việc ra quyết định.

II. CÁC CƠ SỞ LÝ THUYẾT

1. Giải thuật lan truyền ngược

Để huấn luyện mạng neural ta cung cấp cho nó một bộ huấn luyện và cho phép nó học bằng cách điều chỉnh trọng số của các liên kết mạng. Một tập huấn luyện là một tập hợp các mẫu huấn luyện.

- **Training Set = Set of traing samples**

Một mẫu đào tạo là một cặp của một vector đầu vào và một mẫu vector đầu ra mong muốn. Trong trường hợp đào tạo không có giám sát, các vector đầu ra nên được để null. Chiều dài của vector đầu vào nên được tương tự như số neuron các neural trong lớp đầu vào, và độ dài vector đầu ra nên được bằng số neuron trong lớp đầu ra.

- **Training Samples = (input vector, desired vector)**

Thuật toán Backpropagation là một thuật toán giám sát thường được sử dụng để huấn luyện các mạng feed-forward. Nó được giới thiệu lần đầu tiên bởi Paul Werbos trong cuốn sách “The Roots của Backpropagation”. Ý tưởng cơ bản là xác định mạng neural hoạt động như thế nào với đầu vào mẫu, so sánh khác nhau như thế nào giữa các hành vi mong muốn và sau đó điều chỉnh trọng số của các liên kết để giảm thiểu sự khác biệt. Quá trình này lặp đi lặp lại cho tất cả các mẫu đào tạo trong nhiều lần thiết lập để đảm bảo huấn luyện phù hợp.

Việc huấn luyện mạng MLP bởi thuật toán lan truyền ngược sai số bao gồm 2 quá trình: Quá trình truyền thẳng và quá trình truyền ngược. Trong quá trình truyền thẳng, các vector đầu vào sẽ được cung cấp cho các neural của mạng và tín hiệu sẽ được lan truyền lần lượt trên từng lớp mạng. Cuối cùng ta sẽ tính được một tập các đầu ra thực sự của mạng. Trong suốt quá trình truyền thẳng, tất cả các trọng số liên kết của mạng đều cố định. Ngược lại, trong quá trình truyền ngược, tất cả các trọng số liên kết đó sẽ được hiệu chỉnh theo các luật hiệu chỉnh trong số. Sai số của mạng sẽ được đo bằng độ sai lệch giữa đầu ra thu được với các giá trị mục tiêu tương ứng. Các sai số này sau đó sẽ được lan truyền ngược lần lượt trên các lớp mạng (từ lớp cuối cùng đến lớp đầu tiên). Các trọng số liên kết sẽ được hiệu chỉnh sao cho các đầu ra thực sự của mạng càng gần với các giá trị mục tiêu càng tốt.

2. Learning Rate

Learning Rate là một trong những thông số mà điều chỉnh việc làm thế nào để một mạng neural học nhanh và làm thế nào để việc huấn luyện hiệu quả.

Hãy xem xét một neural mà đang trải qua quá trình học tập. Giả định rằng trọng số của một số liên kết trong mạng một phần được đào tạo là 0,3. Khi mạng được giới thiệu một mẫu huấn luyện mới, thuật toán huấn luyện yêu cầu các liên kết thay đổi trọng số của nó đến 0,7 để nó có thể học các mẫu mới phù hợp. Nếu chúng ta cập nhật trọng số ngay lập tức, các mạng neural chắc chắn sẽ học các mẫu mới, nhưng nó có xu hướng quên đi tất cả các mẫu nó đã học trước đó. Điều này là do trọng số hiện tại (0,3) là kết quả của tất cả việc học mà nó đã trải qua cho đến nay.

Vì vậy chúng ta không trực tiếp thay đổi trọng số tới 0,7. Thay vào đó, chúng ta tăng nó bởi một phần nhỏ (chọn 25%) của sự thay đổi cần thiết. Vì vậy, trọng số liên kết của ta được thay đổi thành 0,4 và chúng ta chuyển sang mẫu đào tạo tiếp theo. Yếu tố này (0,25 trong trường hợp này) được gọi là Learning Rate. Căn cứ theo cách này, tất cả các mẫu huấn luyện được huấn luyện trong một số thứ tự ngẫu nhiên. Khi chu trình đào tạo lặp đi lặp lại nhiều lần, cuối cùng mạng neural học được tất cả các mẫu có hiệu quả.

Learning Rate là một giá trị trong khoảng từ 0 đến 1. Nếu chọn 1 giá trị rất gần bằng không, đòi hỏi một số lượng lớn các chu trình huấn luyện. Điều này làm cho quá trình huấn luyện rất chậm. Mặt khác, nếu learning rất lớn, trọng số khác nhau và độ lệch hàm mục tiêu dao động lớn và mạng đạt đến một trạng thái mà việc huấn luyện diễn ra vô ích.

3. Hàm Active

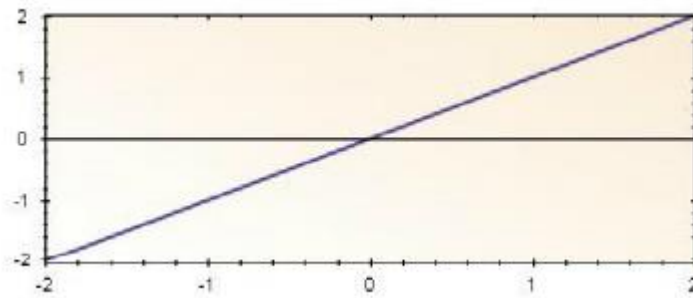
Hàm active trong mạng được xác định là cách để có được đầu ra của neural từ một tập đầu vào dựa trên thuật toán backpropagation.

Các thuật toán backpropagationn yêu cầu một hàm active để thỏa mãn tính liên tục và khả vi. Nó yêu cầu cần có hàm active để dễ dàng tính toán.

Có một số hàm active phổ biến như sau:

- **Hàm đồng nhất:**

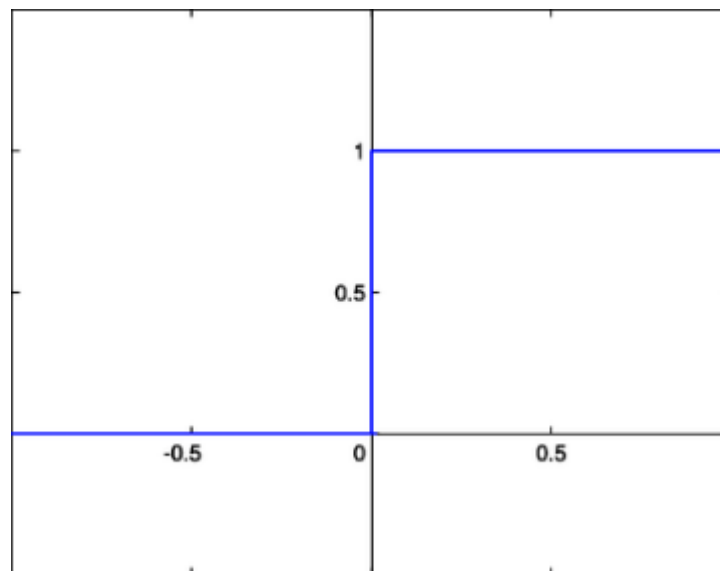
$$g(x) = x$$



Hình 4. Hàm đồng nhất (Identity function)

- **Hàm bước nhị phân:**

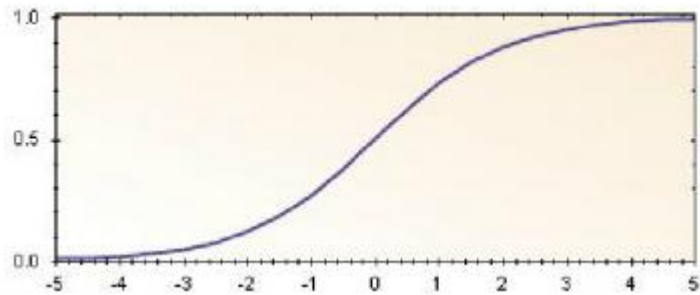
$$g(x) = \begin{cases} 1, & \text{nếu } (x \geq \theta) \\ 0, & \text{nếu } (x < \theta) \end{cases}$$



Hình 5. Hàm bước nhị phân (Binary step function)

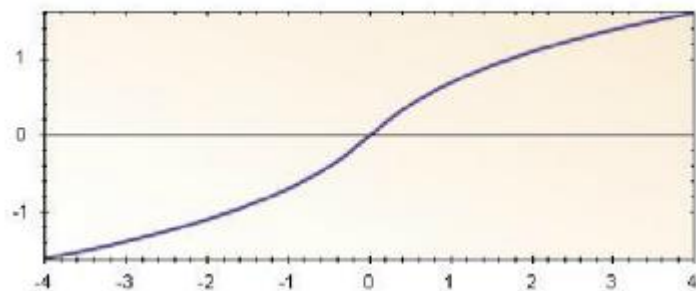
- **Hàm Sigmoid:**

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

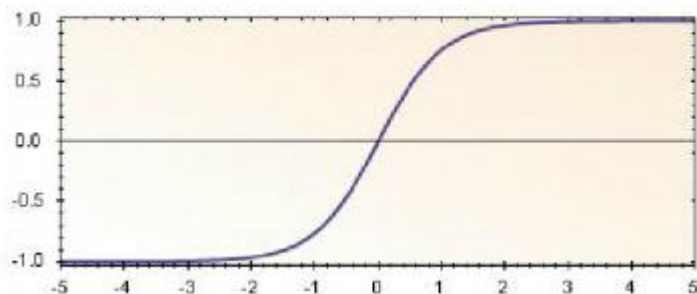


Hình 6. Hàm Sigmoid

- Hàm $y = \text{Log}(1 + |x|)$:

Hình 7. Hàm $y = \text{Log}(1 + |x|)$

- Hàm $y = \text{Tan}(x)$:

Hình 8. Hàm $y = \text{Tan}(x)$

4. Một số gợi ý thiết kế mạng Neural

Thiết kế mạng neural nhân tạo cho một ứng dụng cụ thể liên quan đến việc lựa chọn đúng loại mạng, tìm một số thích hợp các lớp ẩn, phương pháp thích hợp để khởi tạo trọng số, thuật toán học thích hợp, các thể đào tạo, tỷ lệ học tập và số lượng mẫu đào tạo để sử dụng. Hầu hết các thông số này đều phụ thuộc vào ứng dụng mà các mạng neural đang được thiết kế.

Dưới đây là một số hướng dẫn chung về thiết kế mạng neural:

- Số lượng lớp ẩn (trong mạng Backpropagation):

Một mạng lưới Backpropagation không có lớp ẩn không thể thực hiện phân loại không tuyến tính. Vì vậy, một trong những lớp ẩn phải cho là một mạng lưới backpropagation.

Hơn nữa, nó đã được toán học chứng minh rằng một mạng lưới backpropagation với lớp ẩn duy nhất khi đào tạo phù hợp, có thể dùng để xấp xỉ hàm. Vì vậy, lớp ẩn duy nhất là sự lựa chọn tốt nhất trong hầu hết trường hợp.

Có nhiều lớp ẩn tăng tốc quá trình học tập và mạng được đào tạo phù hợp chính xác với các mẫu đào tạo nhưng không thực hiện tốt trên các dữ liệu thử nghiệm. Hiệu ứng này được gọi là overtraining nơi mạng lưới huấn luyện có xu hướng ghi nhớ các mẫu huấn luyện thay vì học tập chúng.

- **Mạng neural ban đầu:**

Khởi tạo đúng trọng số liên kết neural có ảnh hưởng lớn tới việc đào tạo tốc độ cũng như xác định hiệu quả của đào tạo. Thông thường, trọng số được khởi tạo với giá trị ngẫu nhiên từ -0,5 đến +0,5 (giá trị cao có xu hướng kết quả trong khu vực bão hòa sau khi kích hoạt), các giá trị ban đầu nhỏ thì ra các giá trị gần bằng không).

Mạng neural thực hiện quá trình khởi tạo như là một module pluggable. Tùy chỉnh các thuật toán khởi tạo có thể được cắm vào bằng cách thực hiện Initializer giao diện.

- **Số mẫu huấn luyện:**

Quyết định như thế nào để các mẫu đại diện cho các chức năng huấn luyện thực tế là tốt nhất. Thông thường, các lỗi học hơi tăng so với sự gia tăng kích thước của tập huấn luyện, đồng thời ta cũng có thể nhận thấy rằng sự giảm lỗi và mạng thực hiện tốt hơn với dữ liệu thử nghiệm.

Một mối quan hệ giữa kích thước mạng và số lượng tối ưu của mẫu đào tạo có thể được tìm thấy.

III. PHÂN TÍCH, THIẾT KẾ DỮ LIỆU VÀ MẠNG NEURAL

1. Ý tưởng

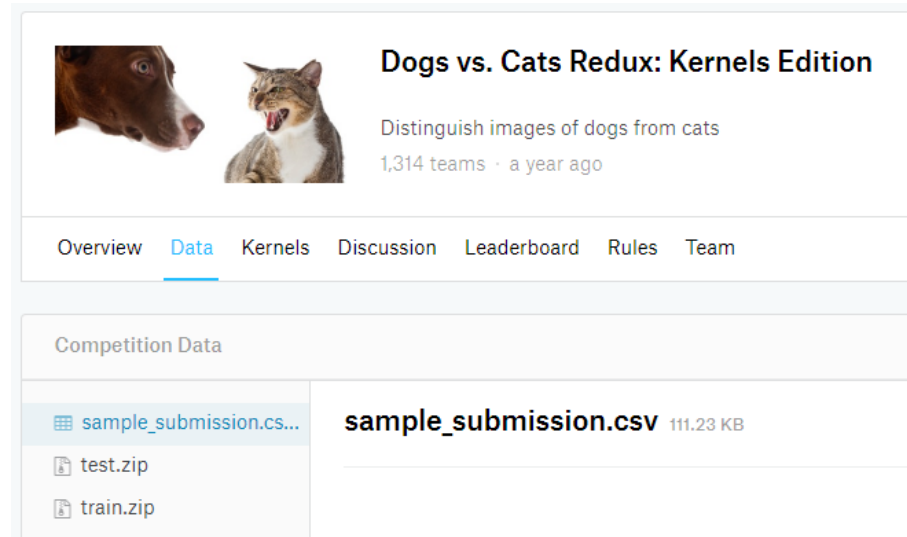
- **Tập dữ liệu:**

- Đặt vấn đề:

Dữ liệu cần dùng để huấn luyện mô hình mạng là các ảnh chó và mèo đã được gán nhãn. Dữ liệu dùng để test là các ảnh chó và mèo chưa được gán nhãn. Số lượng dữ liệu phải đủ lớn và có chất lượng đủ tốt để đảm bảo cho quá trình huấn luyện => Cần tìm nguồn dữ liệu phù hợp.

- Hướng giải quyết:

Kaggle là trang web có hệ thống các cuộc thi, xếp hạng chung, forum kết nối cộng đồng và 1 job board để mọi người có thể tìm kiếm công việc/ứng viên về ngành khoa học dữ liệu (KHDL – data science). Trang web <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data> cung cấp cho ta một nguồn dữ liệu cần thiết như ở trên.



Hình 9. Kaggle

- **Mạng neural:**
 - Đặt vấn đề:

Chọn một ngôn ngữ lập trình để xây dựng một mạng neural cơ bản và huấn luyện cho nó. Xác định các lớp input, các lớp ẩn và lớp output cùng với các hàm active và thuật toán để huấn luyện.

- Hướng giải quyết:

Sử dụng ngôn ngữ Java và công cụ Eclipse để xây dựng mạng neural cho nhận dạng chó mèo. Lớp input sẽ là dữ liệu của 1 sample (ở đây là ảnh) của chó hoặc mèo. Lớp ẩn và lớp output sẽ dùng hàm Sigmoid để kích hoạt dữ liệu đầu ra cho mỗi neural.

2. Phân tích

- **Xử lý ảnh:**
 - Đặt vấn đề:

Máy tính chỉ nhận dạng được dữ liệu của 1 bức ảnh dưới dạng các điểm ảnh gọi là pixel. Mỗi pixel chứa giá trị của 3 màu cơ bản là đỏ,

xanh dương, xanh lục. Mỗi giá trị từ 0-255 => Phải có quá trình xử lý ảnh

- Hướng giải quyết:

Viết 1 chương trình để xử lý ảnh gồm các công đoạn sau: mở file ảnh -> chuyển đổi ảnh về 1 kích thước cố định -> chuyển ảnh về đa mức xám (GrayScale) -> lưu lại giá trị của từng pixel vào 1 mảng -> Lưu giá trị của mảng vào file.txt

- **Mạng neural:**

- Đặt vấn đề:

Cần tạo 1 lớp input bao gồm giá trị của toàn bộ pixel của 1 mẫu ảnh. Ngoài ra còn có thêm 1 giá trị bias.

- Hướng giải quyết:

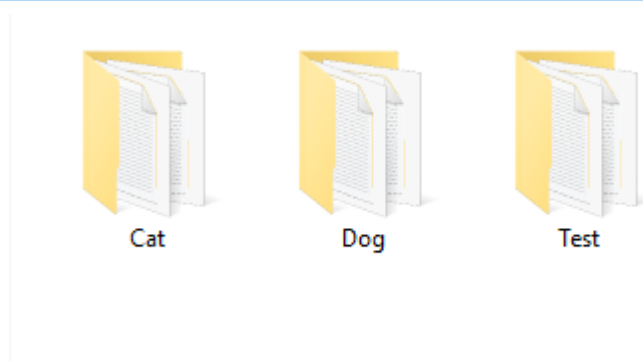
Đọc giá trị từ file.txt và dùng mảng 1 chiều để chứa giá trị cho lớp input. Đối với bias chỉ cần tạo ngẫu nhiên từ 0-1.

3. Thiết kế

- **Tập dữ liệu**

Các folder gồm có: 2 thư mục Cat và Dog sử dụng cho quá trình training và thư mục Test (có gán nhãn) sử dụng cho quá trình testing

E:\Source code\Neural-Network-CatAndDog-Classification-master2\res\data\



Hình 10. Tập dữ liệu

Các file.txt lưu giữ giá trị của `GreyImage[][]` của mỗi bức ảnh với kích thước 100x100 pixel

The image shows a screenshot of a text editor displaying a large grid of numbers. The numbers are arranged in rows and columns, with some cells containing bolded text like 'cat0.txt', 'dog3.txt', and 'cat0.txt'. The grid is composed of numbers ranging from 1 to 100, with some numbers repeated. The text is displayed in a monospaced font, and the background is white.

Hình 11. Dữ liệu file.txt

- **Mô hình mạng neural sẽ sử dụng:**

Sử dụng mô hình mạng neural 3 lớp gồm có: 1 lớp input, 1 lớp ẩn và 1 lớp output.

IV. TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

1. Khởi tạo các class

- **Xử lý ảnh:**

Chương trình xử lý ảnh với thuật toán cơ bản như hình


```

    try {
        bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an 2\\train\\cat."+i+".jpg"));
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    bi = Resize(bi, 100, 100);
    int r=0;
    int g=0;
    int b=0;
    int avg=0;
    int[][] grayImg = new int[100][100];
    for (int y = 0; y < 100; y++) {
        for (int x = 0; x < 100; x++) {
            int p = bi.getRGB(x, y);
            r = (p >> 16) & 0xff;
            g = (p >> 8) & 0xff;
            b = p & 0xff;
            avg = (r + g + b) / 3;
            grayImg[x][y] = avg;
        }
    }
    File file = new File("C:\\Users\\kubim\\Desktop\\dataset\\cat\\cat"+i+".txt");
    try {
        if (file.createNewFile()) {
            System.out.println("Tao xong file!");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Hình 12. Thuật toán xử lý ảnh

- **Neural:**

Tạo class Neural có các thuộc tính cơ bản như hình:

```

public abstract class Neuron {
    protected int numInputs; // số input vào Neuron == số pixel của ảnh
    protected double[] inputWeights;
    protected ArrayList<Double> inputs;
    protected double output; // [0,1]
    protected double gradient;
    protected double bias;

    Neuron(final int numInputs, ArrayList<Double> inputs, double initialWeightClamp) {
        this.numInputs = numInputs;
        this.inputs = inputs;
        bias = getRandomWeight(initialWeightClamp);

        inputWeights = new double[numInputs];
        for (int i = 0; i < numInputs; i++) {
            inputWeights[i] = getRandomWeight(initialWeightClamp);
        }
    }

    private double getRandomWeight(double initialWeightClamp) {
        return (Math.random() - 0.5);
    }
}

```

Hình 13. Class Neural

- **HiddenNeural:**

Class HiddenNeural kế thừa từ class Neural

```
public class HiddenNeuron extends Neuron {
    HiddenNeuron(final int numInputs, ArrayList<Double> inputs, double initialWeightClamp) {
        super(numInputs, inputs, initialWeightClamp);
    }

    public void computeGradient(final double targetOutput, double weight, double outputGradient) {
        gradient = output * (1 - output) * weight * outputGradient;
    }
}
```

Hình 14. Class HiddenNeural

- **OutputNeural:**

Class OutputNeural cũng kế thừa từ class Neural

```
public class OutputNeuron extends Neuron {
    OutputNeuron(final int numInputs, ArrayList<Double> inputs, double initialWeightClamp) {
        super(numInputs, inputs, initialWeightClamp);
    }

    public void computeGradient(final double targetOutput) {
        gradient = output * (1 - output) * (targetOutput - output);
    }
}
```

Hình 15. Class OutputNeural

- **NeuralNetwork:**

Class NeuralNetwork có các thuộc tính cơ bản như hình:

```
public class NeuralNetwork {
    public static final double LEARNING_ACCURACY = 0.95;
    private static final double CLASSIFICATION_TARGET_DOG = 0.9;
    private static final double CLASSIFICATION_TARGET_CAT = 0.1;
    private static final int NUM_INPUT_NODES = 10000;
    private static final int NUM_HIDDEN_NODES = 200;
    private static final double LEARNING_FACTOR = 0.3;
    private static final double INITIAL_WEIGHT_VALUE_CLAMP = 0.5;

    private final ArrayList<Double> inputLayer = new ArrayList<>();
    private final ArrayList<HiddenNeuron> hiddenLayer = new ArrayList<>();
    private final ArrayList<Double> hiddenLayerOutputs = new ArrayList<>();
    private final OutputNeuron outputLayer = new OutputNeuron(NUM_HIDDEN_NODES, hiddenLayerOutputs, INITIAL_WEIGHT_VALUE_CLAMP);

    {
        for (int i = 0; i < NUM_HIDDEN_NODES; i++) {
            hiddenLayer.add(new HiddenNeuron(NUM_INPUT_NODES, inputLayer, INITIAL_WEIGHT_VALUE_CLAMP));
        }
    }
}
```

Hình 16. Class NeuralNetwork

- **Sigmoid Active Function:**

```

/
protected double sigmoidActivationFunction(double x) {
    return 1.0 / (1.0 + Math.exp(-x)); //Sigmoid activation function
}

```

Hình 17. Sigmoid Function

2. Chương trình hoàn chỉnh

- **Lập trình xử lý ảnh**

- Chương trình:

```

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

import javax.imageio.ImageIO;
import javax.swing.JFrame;

public class Convert extends JFrame {
    BufferedImage bi;

    public static void main(String[] args) {

        PrintCat();
        PrintDog();
        PrintTestNoLabel();
        PrintTestLabel();
        Random rand = new Random();
    }

    public static void PrintCat() {
        BufferedImage bi = null;
        for (int i = 0; i < 12500; i++) {
            try {
                bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an
                2\\train\\cat." + i + ".jpg"));
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            bi = Resize(bi, 100, 100);
            int r = 0;
            int g = 0;
            int b = 0;
            int avg = 0;
            int[][] grayImg = new int[100][100];
            for (int y = 0; y < 100; y++) {
                for (int x = 0; x < 100; x++) {
                    int p = bi.getRGB(x, y);
                    r = (p >> 16) & 0xff;
                    g = (p >> 8) & 0xff;
                    b = p & 0xff;
                    avg = (r + g + b) / 3;
                }
            }
        }
    }

```

```

        grayImg[x][y] = avg;
    }
}
File file = new
File("C:\\Users\\kubim\\Desktop\\dataset\\cat\\cat" + i + ".txt");
try {
    if (file.createNewFile()) {
        System.out.println("Tao xong file!");
    }
} catch (IOException e) {
    e.printStackTrace();
}
// ghi file
try {
    // tao doi tuong luong va lien ket nguon du lieu
    FileWriter fw = new FileWriter(file);
    // tao bo dem ghi file
    BufferedWriter bw = new BufferedWriter(fw);
    for (int j = 0; j < grayImg.length; j++) {
        for (int j2 = 0; j2 < grayImg.length; j2++) {
            bw.write(grayImg[j][j2] + " ");
        }
        bw.write("\n");
    }
    bw.close();
    fw.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
System.out.println("Ghi thanh cong");
}
}

public static void PrintDog() {
    BufferedImage bi = null;
    for (int i = 0; i < 12500; i++) {
        try {
            bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an
            2\\train\\dog." + i + ".jpg"));
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        bi = Resize(bi, 100, 100);
        int r = 0;
        int g = 0;
        int b = 0;
        int avg = 0;
        int[][] grayImg = new int[100][100];
        for (int y = 0; y < 100; y++) {
            for (int x = 0; x < 100; x++) {
                int p = bi.getRGB(x, y);
                r = (p >> 16) & 0xff;
                g = (p >> 8) & 0xff;
                b = p & 0xff;
                avg = (r + g + b) / 3;
                grayImg[x][y] = avg;
            }
        }
    }
}

```

```

    }
    File file = new
    File("C:\\Users\\kubim\\Desktop\\dataset\\dog\\dog" + i + ".txt");
    try {
        if (file.createNewFile()) {
            System.out.println("Tao xong file!");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    // ghi file
    try {
        // tao doi tuong luong va lien ket nguon du lieu
        FileWriter fw = new FileWriter(file);
        // tao bo dem ghi file
        BufferedWriter bw = new BufferedWriter(fw);
        for (int j = 0; j < grayImg.length; j++) {
            for (int j2 = 0; j2 < grayImg.length; j2++) {
                bw.write(grayImg[j][j2] + " ");
            }
            bw.write("\n");
        }
        bw.close();
        fw.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("Ghi thanh cong");
}
}

public static void PrintTestNoLabel() {
    BufferedImage bi = null;
    for (int i = 1; i <= 12500; i++) {
        try {
            bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an
            2\\test\\" + i + ".jpg"));
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        bi = Resize(bi, 100, 100);
        int r = 0;
        int g = 0;
        int b = 0;
        int avg = 0;
        int[][] grayImg = new int[100][100];
        for (int y = 0; y < 100; y++) {
            for (int x = 0; x < 100; x++) {
                int p = bi.getRGB(x, y);
                r = (p >> 16) & 0xff;
                g = (p >> 8) & 0xff;
                b = p & 0xff;
                avg = (r + g + b) / 3;
                grayImg[x][y] = avg;
            }
        }
    }
}

```

```

File file = new
File("C:\\Users\\kubim\\Desktop\\dataset\\testnolabel\\" + i +
".txt");
try {
if (file.createNewFile()) {
System.out.println("Tao xong file!");
}
} catch (IOException e) {
e.printStackTrace();
}
// ghi file
try {
// tao doi tuong luong va lien ket nguon du lieu
FileWriter fw = new FileWriter(file);
// tao bo dem ghi file
BufferedWriter bw = new BufferedWriter(fw);
for (int j = 0; j < grayImg.length; j++) {
for (int j2 = 0; j2 < grayImg.length; j2++) {
bw.write(grayImg[j][j2] + " ");
}
bw.write("\n");
}
bw.close();
fw.close();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
System.out.println("Ghi thanh cong");
}
}

public static void PrintTestLabel() {
BufferedImage bi = null;
for (int i = 0; i < 12500; i++) {
try {
// System.out.println("D:\\Ku Bim\\Hoc ky VI\\Do an
// 2\\train\\cat."+i+".jpg");
bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an
2\\train\\cat." + i + ".jpg"));
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
bi = Resize(bi, 100, 100);
int r = 0;
int g = 0;
int b = 0;
int avg = 0;
int[][] grayImg = new int[100][100];
for (int y = 0; y < 100; y++) {
for (int x = 0; x < 100; x++) {
int p = bi.getRGB(x, y);
r = (p >> 16) & 0xff;
g = (p >> 8) & 0xff;
b = p & 0xff;
avg = (r + g + b) / 3;
grayImg[x][y] = avg;
}
}
}

```

```

    }
    File file = new File("C:\\Users\\kubim\\Desktop\\dataset\\test\\"
+ i + "_cat.txt");
    try {
        if (file.createNewFile()) {
            System.out.println("Tao xong file!");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    // ghi file
    try {
        // tao doi tuong luong va lien ket nguon du lieu
        FileWriter fw = new FileWriter(file);
        // tao bo dem ghi file
        BufferedWriter bw = new BufferedWriter(fw);
        for (int j = 0; j < grayImg.length; j++) {
            for (int j2 = 0; j2 < grayImg.length; j2++) {
                bw.write(grayImg[j][j2] + " ");
            }
            bw.write("\n");
        }
        bw.close();
        fw.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("Ghi thanh cong");
}
for (int i = 0; i < 12500; i++) {
    try {
        bi = ImageIO.read(new File("D:\\Ku Bim\\Hoc ky VI\\Do an
2\\train\\dog." + i + ".jpg"));
    } catch (IOException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    bi = Resize(bi, 100, 100);
    int r = 0;
    int g = 0;
    int b = 0;
    int avg = 0;
    int[][] grayImg = new int[100][100];
    for (int y = 0; y < 100; y++) {
        for (int x = 0; x < 100; x++) {
            int p = bi.getRGB(x, y);
            r = (p >> 16) & 0xff;
            g = (p >> 8) & 0xff;
            b = p & 0xff;
            avg = (r + g + b) / 3;
            grayImg[x][y] = avg;
        }
    }
    File file = new File("C:\\Users\\kubim\\Desktop\\dataset\\test\\"
+ i + "_dog.txt");
    try {
        if (file.createNewFile()) {
            System.out.println("Tao xong file!");
        }
    }

```

```

    }
    } catch (IOException e) {
    e.printStackTrace();
    }
    // ghi file
    try {
    // tạo đối tượng luồng và liên kết nguồn dữ liệu
    FileWriter fw = new FileWriter(file);
    // tạo bộ đệm ghi file
    BufferedWriter bw = new BufferedWriter(fw);
    for (int j = 0; j < grayImg.length; j++) {
    for (int j2 = 0; j2 < grayImg.length; j2++) {
    bw.write(grayImg[j][j2] + " ");
    }
    bw.write("\n");
    }
    bw.close();
    fw.close();
    } catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
    }
    System.out.println("Ghi thành công");
    }
    }

    public static BufferedImage Resize(BufferedImage bi, int width,
    int height) { /*
    * linkImage là tên icon, k kích thước chiều rộng mình
    * muốn, m chiều dài và hàm này trả về giá trị là 1
    * icon.
    */

    return toBufferedImage(bi.getScaledInstance(width, height,
    bi.SCALE_SMOOTH));
    }

    public static BufferedImage toBufferedImage(Image img) {
    if (img instanceof BufferedImage) {
    return (BufferedImage) img;
    }

    // Create a buffered image with transparency
    BufferedImage bimage = new BufferedImage(img.getWidth(null),
    img.getHeight(null), BufferedImage.TYPE_INT_ARGB);

    // Draw the image on to the buffered image
    Graphics2D bGr = bimage.createGraphics();
    bGr.drawImage(img, 0, 0, null);
    bGr.dispose();

    // Return the buffered image
    return bimage;
    }
    }

```


- Mô tả:

Chương trình bao gồm hàm `toBufferedImage()` để chuyển đổi đối tượng thuộc lớp `Image()` thành đối tượng thuộc lớp `BufferedImage`.
Hàm `Resize` để quy định kích thước cố định cho 1 bức ảnh.
Các hàm `PrintCat()`, `PrintDog()`, `PrintTestNoLabel()` và `PrintTestLable` để in ra các file.txt chứa dữ liệu của mỗi hình.

- **Lập trình nhận dạng chó mèo**

- Chương trình:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;

public class Main {
    private NeuralNetwork neuralNetwork = new NeuralNetwork();
    private File DogFolder = new File("res/data/Dog");
    private File CatFolder = new File("res/data/Cat");
    private File testFolder = new File("res/data/Test");

    private ArrayList<File> Testfiles = new ArrayList<>();

    private List<Data> Listdata;

    public static void main(String[] args) {
        Main main = new Main();
        main.train();
        main.test();
        // for (int i = 1; i <= 22; i++) {
        //     main.test(new
        File("C:\\Users\\kubim\\Desktop\\dataset\\testnolabel\\" + i + ".tx
        t"));
        // }
    }

    //Đọc file từ folder ==> Lưu data vào Listdata và Train NN
    //In accuracy sau mỗi lần train
    public void train() {
        Listdata = new ArrayList<>();

        File[] dogs = DogFolder.listFiles();

        File[] cats = CatFolder.listFiles();
        int size = dogs.length > cats.length ? dogs.length :
        cats.length;
        for (int i = 0; i < size; i++) {

            if (i < dogs.length && !dogs[i].isHidden() &&
            dogs[i].isFile())
                Listdata.add(new Data(dogs[i], true));
            if (i < cats.length && !cats[i].isHidden() &&
            cats[i].isFile())
                Listdata.add(new Data(cats[i], false));
        }
    }
}
```

```

        double accuracy;
        for (int i = 0; i < 500; i++) {
            //iterations
            accuracy = neuralNetwork.trainNetwork(Listdata);

            System.out.println("Accuracy: " + accuracy);

            if (accuracy >= NeuralNetwork.LEARNING_ACCURACY)
                break;
        }

        //ghi lại trọng số
        try (FileOutputStream fout = new
FileOutputStream("weights")) {
            neuralNetwork.saveWeights(fout);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    public void test() {
        Testfiles = new ArrayList<>();
        for (File file : testFolder.listFiles()) {
            if (!file.isFile() || file.isHidden())
                continue;
            Testfiles.add(file);
        }

        try (FileInputStream fin = new
FileInputStream("weights")) {
            neuralNetwork.loadWeights(fin);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }

        neuralNetwork.testNetwork(Testfiles);
    }

    public void test(File file) {
        try (FileInputStream fin = new
FileInputStream("weights")) {
            neuralNetwork.loadWeights(fin);
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
        neuralNetwork.testNetwork(file);
    }
}

```

○ Mô tả:

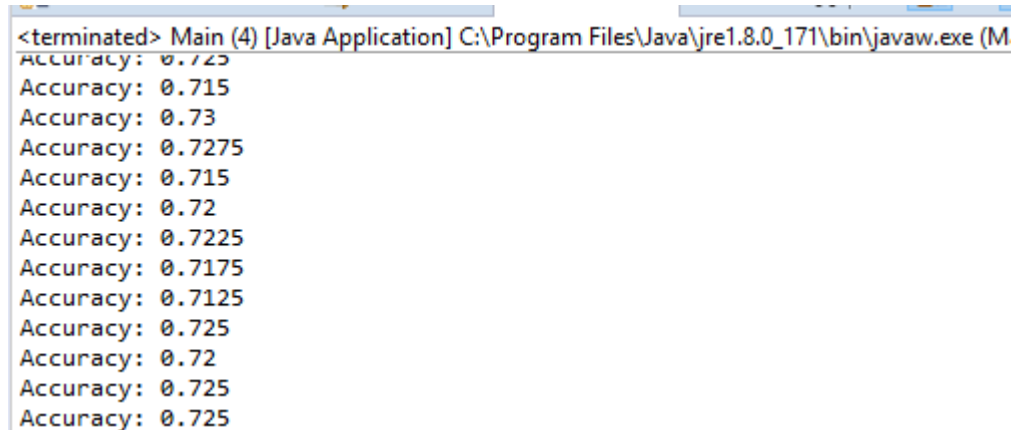
Hàm train() dùng để train dữ liệu, sau khi kết thúc quá trình sẽ in ra 1 file “weight” để lưu các giá trị về trọng số.

Ngoài ra có hàm test() để test các dữ liệu đã được gán nhãn và test(File file) để test dữ liệu mới chưa được gán nhãn.

3. Thực thi và đánh giá kết quả

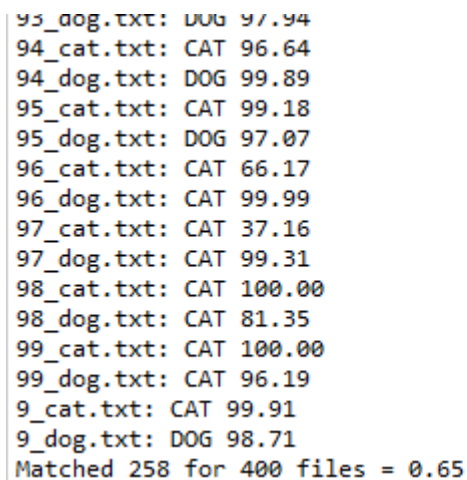
Thực hiện thử quá trình train dữ liệu với 200 bức ảnh mèo được gán nhãn, 200 bức ảnh chó được gán nhãn và test dữ liệu với 400 bức ảnh chó mèo được gán nhãn.

Số node vào của mỗi bức hình là 10000 cùng với số node ẩn là 200 và số vòng lặp là 200.



```
<terminated> Main (4) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (M
Accuracy: 0.725
Accuracy: 0.715
Accuracy: 0.73
Accuracy: 0.7275
Accuracy: 0.715
Accuracy: 0.72
Accuracy: 0.7225
Accuracy: 0.7175
Accuracy: 0.7125
Accuracy: 0.725
Accuracy: 0.72
Accuracy: 0.725
Accuracy: 0.725
```

Hình 18. Kết quả train



```
93_dog.txt: DOG 97.94
94_cat.txt: CAT 96.64
94_dog.txt: DOG 99.89
95_cat.txt: CAT 99.18
95_dog.txt: DOG 97.07
96_cat.txt: CAT 66.17
96_dog.txt: CAT 99.99
97_cat.txt: CAT 37.16
97_dog.txt: CAT 99.31
98_cat.txt: CAT 100.00
98_dog.txt: CAT 81.35
99_cat.txt: CAT 100.00
99_dog.txt: CAT 96.19
9_cat.txt: CAT 99.91
9_dog.txt: DOG 98.71
Matched 258 for 400 files = 0.65
```

Hình 19. Kết quả test

V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

Như vậy, với đề tài đồ án 2: **Xây dựng mạng nơron nhân tạo để nhận dạng chó mèo** em đã thực hiện xây dựng chương trình mô phỏng theo mạng nơron nhân tạo để nhận dạng chó mèo sử dụng ngôn ngữ Java

Với sự nỗ lực trong việc tìm hiểu nghiên cứu các đề tài, bài viết cần thiết cho đề tài, cùng với vận dụng các kiến thức đã học vào công việc thiết

kế, viết chương trình, và sự giúp đỡ, hướng dẫn nhiệt tình của thầy giáo TS. Phạm Minh Tuấn, em đã hoàn thành được đề tài đồ án 2: **Xây dựng mạng noron để nhận dạng chó mèo**. Trong quá trình thực hiện, lập trình dự án, em gặp phải nhiều khó khăn khác nhau như: do phải nghiên cứu nhiều tài liệu, lý thuyết, ... dẫn đến nhiều chỗ dịch sai, dịch nhầm dẫn đến áp dụng các hàm, câu lệnh bị sai ý nghĩa, cấu trúc..., trong quá trình viết code gặp phải nhiều lỗi phát sinh mà không tìm ngay ra nguyên nhân cần đầu tư thời gian để giải quyết, nhiều thư viện rất khó sử dụng để lập trình... Quá trình viết code cũng gặp phải những khó khăn nhất định tuy nhiên em đã cố gắng giải quyết được vấn đề phát sinh để hoàn thành được đề tài. Em đã hoàn thành nghiên cứu, xây dựng và thiết kế mạng noron để nhận dạng chó mèo sử dụng ngôn ngữ Java trong vòng hơn 4 tháng từ khi nhận đề tài.

Do đây kiến thức còn nhiều hạn chế, em tự thấy đề tài của mình thực hiện được vẫn còn rất nhiều sai sót, khiếm khuyết. Em rất mong được sự ủng hộ và giúp đỡ của thầy giáo để đề tài em thực hiện được hoàn thiện hơn và có thêm nhiều cải tiến đáng kể và ứng dụng tốt hơn vào thực tiễn.

2. Hướng phát triển

Sử dụng mạng Noron tích chập để trích xuất các đặc trưng của ảnh để có thể nhận dạng tốt hơn.

Đầu tư thời gian cho quá trình train và test nhiều hơn để tăng độ chính xác.

TÀI LIỆU THAM KHẢO

- [1] <https://techmaster.vn/>
- [2] <http://ramok.tech/2017/11/29/digit-recognizer-with-neural-networks/>
- [3] <http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-networks>
- [4] <https://becominghuman.ai/making-a-simple-neural-network-classification-2449da88c77e>
- [5] <https://github.com/>
- [6] <https://machinelearningcoban.com/>
- [7] https://vi.wikipedia.org/wiki/M%E1%BA%A1ng_n%C6%A1-ron_nh%C3%A2n_t%E1%BA%A1o