TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ

ĐỒ ÁN TỐT NGHIỆP

NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH
HỆ ĐÀO TẠO CHẤT LƯỢNG CAO

NGHIÊN CỨU VÀ TỐI ƯU MÔ HÌNH MÁY HỌC TRÊN KHUNG SƯỜN THIẾT BỊ NHÚNG

SVTH: NGUYỄN ĐỨC THỊNH
MSSV: 20119291

TP. HỒ CHÍ MINH – 07/2024

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ

ĐỒ ÁN TỐT NGHIỆP

NGÀNH CÔNG NGHỆ KỸ THUẬT MÁY TÍNH

NGHIÊN CỨU VÀ TỐI ƯU MÔ HÌNH MÁY HỌC TRÊN KHUNG SƯỜN THIẾT BỊ NHÚNG

SVTH: NGUYỄN ĐỨC THỊNH
MSSV: 20119291

GVHD: PGS.TS VÕ MINH HUÂN

TP. HỒ CHÍ MINH – 07/2024

ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐIỆN – ĐIỆN TỬ
Bộ Môn: KTMT - VT

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc
----*----

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên Sinh viên: Nguyễn Đức Thịnh MSSV: 20119291

Ngành: Công nghệ Kỹ thuật Máy tính

Tên đề tài: Nghiên cứu và tối ưu mô hình máy học trên khung sườn thiết bị nhúng

Giáo viên hướng dẫn: PGS.TS Võ Minh Huân

NHẬN XÉT

1. Về nội dung đề tài & khối lượng thực hiện (*khả năng ứng dụng, tính mới, sáng tạo, mức độ đóng góp của sinh viên, ...):*

   Đề tài có tính ứng dụng cao trong việc tích hợp máy học vào thiết bị nhúng, giúp nâng cao hiệu suất và mở rộng tính năng cho các thiết bị. Sinh viên đã thể hiện sự sáng tạo và tinh thần đóng góp vào lĩnh vực này.

2. Hình thức trình bày quyển báo cáo (*Văn phong, trích dẫn tài liệu tham khảo, chất lượng các hình ảnh, bảng biểu, tỷ lệ trùng lắp, ...*):

   Báo cáo trình bày rõ ràng, khoa học và trích dẫn tài liệu đầy đủ. Hình ảnh và bảng biểu có chất lượng tốt, minh họa rõ ràng cho nội dung nghiên cứu.

3. Những hạn chế cần chỉnh sửa, bổ sung:
4. Đề xuất của GVHD (*Đồng ý cho bảo vệ, đề nghị chỉnh sửa để được bảo vệ, không đồng cho bảo vệ*)

   Đồng ý cho bảo vệ

TP. Hồ Chí Minh, ngày 14 tháng 06 năm 2024

GIẢNG VIÊN HƯỚNG DẪN

Võ Minh Huân

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐIỆN – ĐIỆN TỬ

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc Lập – Tự Do – Hạnh Phúc
****

BẢN GIẢI TRÌNH CHỈNH SỬA BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

1. Tên đề tài: Nghiên cứu và tối ưu mô hình máy học trên khung sườn thiết bị nhúng
2. Họ tên sinh viên: Nguyễn Đức Thịnh MSSV: 20119291
3. Ngành: Công nghệ Kỹ thuật Máy tính
4. GVHD: PGS.TS Võ Minh Huân
5. Tổng hợp các yêu cầu chỉnh sửa báo cáo ĐATN của hội đồng:
   - Giải thích độ chính xác suy giảm trong hình 4.5
   - Thuật ngữ "khung sườn" nên được điều chỉnh lại nếu từ gốc là "platform".
   - Cần nêu rõ phương pháp đo lường thời gian thực thi. Vì thời gian thực thi bị ảnh hưởng bởi nhiều yếu tố khách quan không kiểm soát được như thời gian hệ điều hành lập lịch, thời gian truyền dẫn giữa CPU và MEM, thời gian truy cập ngoại vi. Các yếu tố này cần được trình bày rõ.
6. Giải trình chỉnh sửa báo cáo ĐATN

TT	Nội dung góp ý của HĐ	Kết quả chỉnh sửa bổ sung
1	Giải thích độ chính xác suy giảm trong hình 4.5	Đã bổ sung giải thích tại mục 4.3 trang 67
2	Nêu rõ phương pháp đo lường thời gian thực thi	Đã nêu phương pháp đo lường tại mục 2.8 trang 29

GVHD
(Ký tên)

Võ Minh Huân

Sinh viên thực hiện ĐATN
(Ký tên)

Nguyễn Đức Thịnh

LỜI CẢM ƠN

Trong quá trình nghiên cứu và thực hiện khóa luận tốt nghiệp, chúng em đã được tiếp thu và mở rộng thêm nhiều kiến thức mới cũng như phát triển nhiều kỹ năng cùng những trải nghiệm quý báu. Đặc biệt, chúng em xin gửi lời cảm ơn chân thành tới PGS.TS Võ Minh Huân, người thầy đã hỗ trợ chúng em từ những bước đầu tiên cho đến khi hoàn thành khóa luận này. Thầy đã đưa ra những góp ý quý giá giúp chúng em khai phóng tiềm năng của bản thân và định hướng đúng đắn, góp phần vào thành quả đạt được của khóa luận. Chúng em trân trọng những đóng góp mang tính chuyên môn cao của thầy, được đúc kết qua nhiều năm kinh nghiệm làm việc. Thầy luôn sẵn lòng giải đáp thắc mắc và đóng góp ý kiến để chúng em hoàn thiện khóa luận đúng hạn.

Chúng em cũng xin bày tỏ lòng biết ơn tới các thầy, cô trong khoa Điện – Điện tử và ngành Công Nghệ Kỹ Thuật Máy Tính. Trong suốt bốn năm học đại học, các thầy cô đã truyền đạt cho chúng em những kiến thức và kinh nghiệm quý báu, giúp chúng em có nền tảng kiến thức vững chắc để thực hiện khóa luận này. Ngoài ra, chúng em cảm ơn hội đồng đã dành thời gian để đánh giá và trao đổi với chúng em về kết quả của khóa luận này. Những ý kiến, đề xuất và nhận xét từ các thầy, cô trong hội đồng sẽ giúp chúng em nâng cao chất lượng nghiên cứu và hoàn thiện hơn. Chúng em trân trọng sự đánh giá chuyên môn từ các thầy cô và sẽ lấy đó làm kinh nghiệm quý báu để phát triển trong tương lai.

LỜI CAM ĐOAN

Nhóm sinh viên Nguyễn Đức Thịnh thực hiện đề tài "Nghiên cứu và tối ưu mô hình máy học trên khung sườn thiết bị nhúng" dưới dự hướng dẫn của thầy Võ Minh Huân xin cam đoan các nội dung như sau:

1. Sản phẩm của Đồ án tốt nghiệp là do nhóm sinh viên Nguyễn Đức Thịnh thực hiện, không mượn, thuê, mua từ người khác.
2. Quyển báo cáo Đồ án tốt nghiệp là do nhóm sinh viên Nguyễn Đức Thịnh tự viết, tỷ lệ trùng lắp là 33% các nội dung tham khảo đã được trích dẫn đầy đủ
3. Kết quả thực hiện trong quyển báo cáo bao gồm hình ảnh, độ chính xác của mô hình là hoàn toàn đúng với mô hình, phần cứng nhóm đã thực hiện.

Nhóm sinh viên cam đoan các nội dung trên là hoàn toàn chính xác và chịu trách nhiệm hoàn toàn với những cam đoan trên.

Sinh viên thực hiện đồ án tốt nghiệp

(ký và ghi rõ họ tên)

Nguyễn Đức Thịnh

TÓM TẮT

Xây dựng và triển khai mô hình trí thông minh nhân tạo trên các thiết bị nhúng ngày càng phổ biến, mang lại nhiều giá trị trong việc giảm độ trễ, tận dụng hiệu quả băng thông, cải thiện bảo mật dữ liệu, tăng cường quyền riêng tư và giảm chi phí cho người sử dụng. Tuy nhiên, công việc này đưa ra nhiều thách về độ chính xác, tốc độ xử lý, tài nguyên hệ thống và kích thước mô hình. Bên cạnh đó, các thiết bị nhúng cũng cần có một cơ chế quản lý trạng thái hệ thống để tối ưu hóa việc triển khai mô hình trên thiết bị nhúng.

Vì những lý do trên, đề tài "Nghiên cứu và tối ưu mô hình máy học trên khung sườn thiết bị nhúng" tập trung vào việc nghiên cứu và phát triển các mô hình học máy để giải quyết bài toán nhận diện hình ảnh, cụ thể là trên bộ dữ liệu MNIST. Ba mô hình chính được triển khai và đánh giá gồm: Convolutional Neural Network (CNN), Binary Neural Network (BNN) và Linear Support Vector Machine (LSVM) kết hợp với thuật toán K-Means.

CNN được lựa chọn vì khả năng trích xuất đặc trưng mạnh mẽ, đặc biệt hiệu quả trong các bài toán nhận diện hình ảnh. Kết quả thử nghiệm cho thấy CNN đạt độ chính xác lên đến 99% trên bộ dữ liệu MNIST, mặc dù yêu cầu tài nguyên tính toán lớn.

BNN mang lại giải pháp nhẹ hơn, thích hợp cho các thiết bị có tài nguyên hạn chế. Dù độ chính xác của BNN chỉ đạt khoảng 85-89%, nhưng nó vẫn đảm bảo hiệu suất chấp nhận được trong các điều kiện giới hạn tài nguyên.

LSVM kết hợp với K-Means là một giải pháp trung gian, với độ chính xác khoảng 81%. Mô hình này sử dụng K-Means để phân cụm và tăng số lượng mẫu dữ liệu trước khi áp dụng LSVM, giúp tăng tốc độ xử lý và cải thiện độ chính xác so với LSVM đơn thuần.

Ngoài việc đánh giá và so sánh các mô hình, đề tài còn phát triển một framework để triển khai và quản lý các mô hình này một cách hiệu quả. Framework cho phép xử lý ảnh nhanh chóng, với thời gian xử lý trung bình khoảng 350ms mỗi ảnh. Khả năng chuyển đổi linh hoạt giữa các mô hình giúp đáp ứng nhu cầu đa dạng của người dùng.

Kết luận từ đề tài cho thấy, CNN là mô hình hiệu quả nhất với độ chính xác cao nhất, trong khi BNN và LSVM + K-Means là các giải pháp thay thế phù hợp cho các thiết bị có tài nguyên hạn chế. Tuy nhiên, cả ba mô hình đều có những hạn chế và cần được tối ưu hóa thêm để cải thiện hiệu suất và khả năng ứng dụng trong các bài toán phức tạp hơn.

Hướng phát triển của đề tài bao gồm nâng cao độ chính xác của các mô hình, tối ưu hóa tài nguyên, đa dạng hóa các loại mô hình và cải thiện framework để tăng tốc độ xử lý và quản lý tài nguyên hiệu quả hơn. Ngoài ra, việc nâng cấp giao diện người dùng cũng được đề xuất để nâng cao trải nghiệm và hiệu quả sử dụng của người dùng.

ABSTRACT

Building and deploying artificial intelligence models on embedded devices is becoming increasingly popular, offering numerous benefits such as reduced latency, efficient bandwidth utilization, improved data security, enhanced privacy, and reduced costs for users. However, this task presents several challenges, including accuracy, processing speed, system resource constraints, and model size. Additionally, embedded devices require a system state management mechanism to optimize model deployment on these devices.

For these reasons, the topic “Research and optimize machine learning models on embedded device frameworks” focuses on researching and developing machine learning models to solve the image recognition problem, specifically using the MNIST dataset. Three main models are implemented and evaluated: Convolutional Neural Network (CNN), Binary Neural Network (BNN), and Linear Support Vector Machine (LSVM) combined with the K-Means algorithm.

CNN is chosen for its powerful feature extraction capabilities, making it particularly effective in image recognition tasks. Experimental results show that CNN achieves up to 99% accuracy on the MNIST dataset, although it requires significant computational resources.

BNN offers a lighter solution, suitable for devices with limited resources. While BNN’s accuracy ranges from 85-89%, it provides acceptable performance under resource-constrained conditions.

LSVM combined with K-Means serves as an intermediate solution, with an accuracy of about 81%. This model uses K-Means for clustering and increasing the number of data samples before applying LSVM, which helps speed up processing and improve accuracy compared to using LSVM alone.

In addition to evaluating and comparing the models, this project also develops a framework to efficiently deploy and manage these models. The framework allows for quick image processing, with an average processing time

of about 350ms per image. The flexible switching capability between models helps meet diverse user needs.

The conclusion of the project shows that CNN is the most effective model with the highest accuracy, while BNN and LSVM + K-Means are suitable alternatives for devices with limited resources. However, all three models have limitations and need further optimization to improve performance and applicability in more complex tasks.

Future directions for this project include enhancing model accuracy, optimizing resources, diversifying model types, and improving the framework to increase processing speed and resource management efficiency. Additionally, upgrading the user interface is suggested to enhance user experience and effectiveness.

MỤC LỤC

DANH MỤC HÌNH XVI
DANH MỤC BẢNG XIX
CÁC TỪ VIẾT TẮT XX

CHƯƠNG 1 TỔNG QUAN VỀ ĐỀ TÀI **1**

1.1 ĐẶT VẤN ĐỀ 1
1.2 MỤC TIÊU ĐỀ TÀI 6
1.3 GIỚI HẠN ĐỀ TÀI 6
1.4 PHƯƠNG PHÁP NGHIÊN CỨU 7
1.5 ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU 7
1.6 BỐ CỤC QUYỂN BÁO CÁO 8

CHƯƠNG 2 CƠ SỞ LÝ THUYẾT **9**

2.1 MÔ HÌNH MẠNG NƠ-RON 9
2.2 MÔ HÌNH MẠNG CNN 10
2.2.1 Sơ lược về mạng CNN 10
2.2.2 Kiến trúc CNN 10
2.3 MÔ HÌNH MẠNG BNN 13
2.3.1 Lan truyền xuôi (Forward Propagation) 13
2.3.2 Lan truyền ngược (Backward Propagation) 14
2.4 MÔ HÌNH SVM 15
2.4.1 Sơ lược về SVM 15
2.4.2 Các khái niệm chính 16
2.4.3 Mô hình thuật toán SVM 18
2.5 THUẬT TOÁN K-MEANS 20
2.5.1 Lý thuyết 20
2.5.2 Thuật toán phân cụm 21
2.6 GIẢI THUẬT LƯU MÔ HÌNH NHỊ PHÂN 24
2.6.1 BitArray 24
2.6.2 Mã hóa 25

2.6.3 Giải mã 26
2.7 PHƯƠNG PHÁP ĐO TÀI NGUYÊN SỬ DỤNG CỦA MÔ HÌNH 27
2.7.1 Memory profiler 27
2.7.2 Cách thực hiện 27
2.8 PHƯƠNG PHÁP ĐO THỜI GIAN THỰC THI 29
2.8.1 Time 29
2.8.2 Cách thực hiện 30
2.9 FRAMEWORK QUẢN LÝ HỆ THỐNG NHÚNG 32
2.9.1 Lý thuyết về framework 32
2.9.2 Sơ lược về IPC 33
2.9.3 Sơ lược về FSM 35
2.10 JETSON NANO 36
2.10.1 Giới thiệu về Jetson Nano 36
2.10.2 So sánh Jetson Nano và Raspberry Pi 37
CHƯƠNG 3XÂY DỰNG MÔ HÌNH VÀ THIẾT KẾ HỆ THỐNG 39
3.1 MÔ HÌNH CNN 39
3.2 KHẢO SÁT MÔ HÌNH BNN 40
3.2.1 Sơ đồ khối 40
3.2.2 Mã giả 44
3.3 KHẢO SÁT MÔ HÌNH LSVM KẾT HỢP VỚI K-MEANS 50
3.3.1 Sơ đồ khối 50
3.3.2 Mã giả 53
3.4 THIẾT KẾ FRAMEWORK 55
3.4.1 FSM 55
3.5 PHƯƠNG PHÁP ĐÁNH GIÁ 57
CHƯƠNG 4KẾT QUẢ 59
4.1 TIÊU CHÍ ĐÁNH GIÁ 59
4.2 ĐÁNH GIÁ MÔ HÌNH BNN 59
4.3 ĐÁNH GIÁ MÔ HÌNH LSVM KẾT HỢP VỚI K-MEANS 65

4.4 SO SÁNH KẾT QUẢ GIỮA CÁC MÔ HÌNH 70
4.5 SO SÁNH TÀI NGUYÊN SỬ DỤNG GIỮA CÁC MÔ HÌNH 71
4.6 ĐÁNH GIÁ CÁC MÔ HÌNH TRÊN JETSON NANO 75
4.7 GIAO DIỆN FRAMEWORK 76
CHƯƠNG 5KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 78
5.1 KẾT LUẬN 78
5.1.1 Kết luận về mô hình 78
5.1.2 Kết luận về framework 80
5.2 HẠN CHẾ 81
5.2.1 Hạn chế về mô hình 81
5.2.2 Hạn chế về framework 81
5.3 HƯỚNG PHÁT TRIỂN 82
5.3.1 Hướng phát triển mô hình 82
5.3.2 Hướng phát triển framework 82
TÀI LIỆU THAM KHẢO 83

DANH MỤC HÌNH

Hình 1-1 Ứng dụng mô hình AI trên phần cứng..... 1
Hình 1-2 Biểu đồ biểu diễn độ chính xác theo sự phức tạp của mô hình thuật toán[2]. 2
Hình 2-1 Mô hình mạng nơ-ron 9
Hình 2-2 Kiến trúc mạng CNN cơ bản [11]. 11
Hình 2-3 Lớp convolutional của mô hình CNN [11]. 11
Hình 2-4 Hình tượng hóa lớp Pooling với average pooling và max pooling [12]. 12
Hình 2-5 Đặc điểm mô hình BNN so với CNN 13
Hình 2-6 So sánh hàm kích hoạt của mô hình BNN với CNN 14
Hình 2-7 Xử lý tính toán trong lan truyền ngược 15
Hình 2-8 Hình tượng hóa SVM 16
Hình 2-9 Kernel của SVM phi tuyến 17
Hình 2-10 Biểu diễn siêu phẳng của SVM trong không gian 18
Hình 2-11 Biểu diễn lề của siêu mặt trong không gian 19
Hình 2-12 Tính toán khoảng cách điểm đến siêu phẳng 20
Hình 2-13 Biểu diễn tâm cụm với K-Means [17]. 21
Hình 2-14 Biểu đồ distortion tương ứng với số cụm trong thuật toán elbow 22
Hình 2-15 Biểu diễn tính Sihouette trong không gian 23
Hình 2-16 Quá trình mã hóa BitArray 25
Hình 2-17 Quá trình giải mã BitArray 26
Hình 2-18 Lệnh cài thư viện memory profiler 27
Hình 2-19 Sử dụng thư viện memory profiler trong chương trình 28
Hình 2-20 Lệnh đo tài nguyên sử dụng của hàm 28
Hình 2-21 Lệnh thay thế đo tài nguyên sử dụng của hàm 28
Hình 2-22 Kết quả đo đạt tài nguyên sử dụng 28
Hình 2-23 file lưu dữ liệu tài nguyên sử dụng 29
Hình 2-24 Lệnh vẽ biểu đồ tài nguyên sử dụng 29
Hình 2-25 Đặt mốc thời gian bắt đầu đo 30

Hình 2-26 Gọi hàm cần đo và đặt mốc kết thúc thời gian đo 31
Hình 2-27 Tính toán thời gian thực thi 31
Hình 2-28 Mô hình FrameWork 32
Hình 2-29 Kiến trúc máy tính 33
Hình 2-30 Mô hình hàng đợi tin nhắn 34
Hình 2-31 Mô hình trạng thái của trạm sạc xe điện 35
Hình 2-32 Nguyên lý hoạt động của FSM 36
Hình 2-33 Board Jetson Nano 37
Hình 3-1 Mô hình CNN 39
Hình 3-2 Sơ đồ khối mô hình BNN 41
Hình 3-3 Hiệu chỉnh kích thước ảnh 44
Hình 3-4 Mã hóa và nhị phân hóa nhãn 45
Hình 3-5 Mô hình phân cụm với K-Means 50
Hình 3-6 Sơ đồ khối mô hình LSVM 51
Hình 3-7 Quá trình XNOR popcount 52
Hình 3-8 Biểu đồ stateflow của FSM 56
Hình 3-9 Khai báo trạng thái, sự kiện, hành động của FSM 56
Hình 4-1 Biểu diễn độ chính xác và mất mát theo của mô hình BNN 60
Hình 4-2 Biểu diễn độ chính xác và mất mát theo epoch của mô hình BNN+BitArray 62
Hình 4-3 Biểu diễn độ chính xác và mất mát theo epoch của mô hình BNN+K-Means 64
Hình 4-4 Biểu diễn lỗi theo epoch của mô hình LSVM+K-Means 66
Hình 4-5 Biểu diễn độ chính xác theo epoch của mô hình LSVM+K-Means 67
Hình 4-6 Biểu diễn giá trị ngưỡng theo epoch của mô hình LSVM+K-Means 68
Hình 4-7 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình CNN 72
Hình 4-8 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN 72

Hình 4-9 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN+BitArray 73
Hình 4-10 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN+K-Means 73
Hình 4-11 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình SVM 74
Hình 4-12 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình LSVM+K-Means 75
Hình 4-13 Giao diện của FrameWork 76
Hình 4-14 Quá trình chuyển đổi trạng thái trong FSM 77

DANH MỤC BẢNG

Bảng 1-1 Hiệu quả của các phương pháp lượng tử hóa với độ chính xác của ResNet-18 được lượng tử hóa trên ImageNet 2
Bảng 1-2 So sánh đặc điểm giữa CNN, TNN và BNN 3
Bảng 1-3 So sánh một số phương pháp tiếp cận đối với mạng nơ-ron 4
Bảng 2-1 So sánh phương pháp Elbow và phương pháp Silhouette 24
Bảng 2-2 So sánh Jetson Nano Developer Kit A02 với Raspberry Pi 4 Model B 38
Bảng 4-1 Thông số huấn luyện mô hình BNN 59
Bảng 4-2 So sánh kết quả giữa mô hình CNN và BNN 60
Bảng 4-3 Thông số huấn luyện mô hình BNN+BitArray 61
Bảng 4-4 So sánh kết quả giữa mô hình BNN và BNN+BitArrray 62
Bảng 4-5 Thông số huấn luyện mô hình BNN+K-Means 63
Bảng 4-6 So sánh kết quả giữa mô hình BNN+BitArrray và BNN+K-Means 64
Bảng 4-7 Thông số huấn luyện mô hình LSVM+K-Means 65
Bảng 4-8 Kết quả đạt được của mô hình LSVM+K-Means 69
Bảng 4-9 So sánh kết quả của các mô hình 70
Bảng 4-10 Sử dụng tài nguyên bộ nhớ của các mô hình 71
Bảng 4-11 So sánh độ chính xác và thời gian thực thi của các mô hình trên Jetson Nano 75

CÁC TỪ VIẾT TẮT

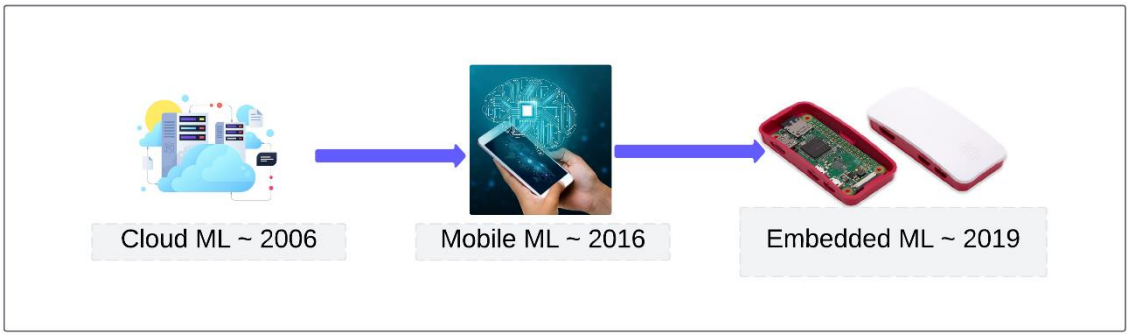
Viết tắt	Mô tả
AI	Artificial Intelligence
IoT	Internet of Things
BNN	Binary Neural Network
CNN	Convolutional Neural Network
TNN	Ternary Neural Network
SVM	Support Vector Machine
LSVM	Linear Support Vector Machine
FSM	Finite State Machine
IPC	Inter-Process Communication
MNIST	Modified National Institute of Standards and Technology
STE	Straight-Through Estimator
SSE	Sum of Squared Errors
ID	Identifier
OS	Operating System
FIFO	First In, First Out
APP	Application
MSE	Mean Squared Error
RAM	Random Access Memory
CPU	Central Processing Unit
GPU	Graphics Processing Unit

CHƯƠNG 1

TỔNG QUAN VỀ ĐỀ TÀI

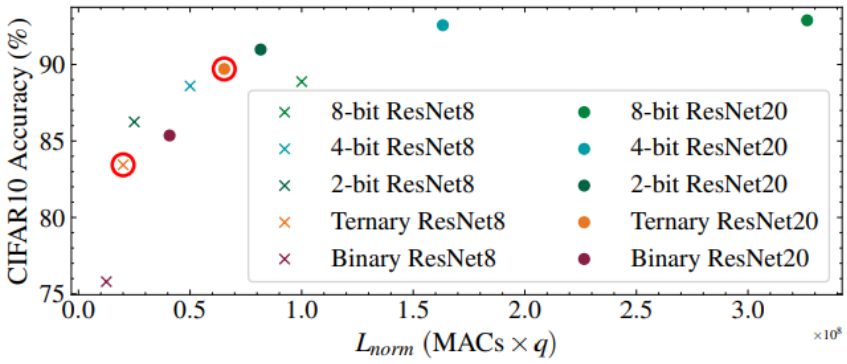
1.1 ĐẶT VẤN ĐỀ

Cuộc cách mạng lần thứ tư xem trí tuệ nhân tạo (AI) và IoT là trọng tâm của quá trình chuyển đổi. Sự chuyển đổi này đã tạo ra loại mạng nơ-ron mới, nhỏ hơn và ưu tiên hiệu quả mô hình. Hình 1-1 mô tả sự phát triển của Cloud ML, Mobile ML và Embedded ML từ 2006 đến 2019. Nhưng hiện nay, các mô hình như CNN mặc dù mang lại độ chính xác cao nhưng việc triển khai mô hình AI trên các thiết bị IoT nhỏ mang lại nhiều thách thức liên quan đến hạn chế về tài nguyên như bộ nhớ, dung lượng lưu trữ và tốc độ xử lý. Do các thiết bị này thường chạy bằng pin, việc tối ưu hóa hiệu quả năng lượng là rất quan trọng để kéo dài thời gian hoạt động [1].



Hình 1-1 Ứng dụng mô hình AI trên phần cứng

Việc giảm kích thước của mô hình đi bằng cách lượng tử hóa số bit thấp sẽ giúp mô hình tính toán nhanh hơn nhưng sẽ làm độ chính xác giảm đi đáng kể. Hình 1-2 cho thấy khi chúng ta biến đổi mô hình ResNet8 từ 8-bit với độ chính xác 89% về nhị phân thì độ chính xác còn lại 76%, chúng ta có sự chênh lệch là 13%.



Hình 1-2 Biểu đồ biểu diễn độ chính xác theo sự phức tạp của mô hình thuật toán[2].

Bảo mật và quyền riêng tư cũng là mối quan tâm lớn, đòi hỏi phải bảo vệ dữ liệu khỏi các cuộc tấn công và xử lý dữ liệu nhạy cảm tại thiết bị biên để giảm thiểu rủi ro. Độ trễ thấp và khả năng kết nối không liên tục là các yếu tố cần cân nhắc để đảm bảo mô hình AI hoạt động hiệu quả ngay cả khi kết nối mạng không ổn định. Khả năng cập nhật mô hình từ xa giúp cải thiện và sửa lỗi mà không cần truy cập vật lý đến thiết bị, trong khi tương thích phần cứng và tích hợp với hệ sinh thái phần mềm hiện có là cần thiết để tạo ra một hệ thống hoàn chỉnh và hiệu quả. Ứng dụng của AI trong IoT bao gồm nhận diện giọng nói, thị giác máy tính và dự đoán bảo trì, đem lại nhiều lợi ích thiết thực cho người dùng.

Trong bài cáo này, nhóm thực hiện đề tài đề cập chính đến mô hình Binary Neural Network (BNN) và mô hình Support Vector Machine (SVM). Hai mô hình này đều gặp phải một số vấn đề chung cần được giải quyết. Cả hai mô hình đều gặp khó khăn trong việc duy trì hiệu suất cao do những hạn chế trong phương pháp nhị phân hóa và tối ưu hóa.

Bảng 1-1 Hiệu quả của các phương pháp lượng tử hóa với độ chính xác của ResNet-18 được lượng tử hóa trên ImageNet [3].

Type	Method	Activation/Weight	Storage Saving	Theoretical Speedup	Top-1 Acc.	Top-5 Acc.
FP32	Original	32-bit Float Point	1×	1×	69.3%	89.2%
INT8	DA-INT8 [49]	8-bit Integer	4×	4×	70.2%	N.A.
TNN	TRQ [23]	Ternary (2-bit)	16×	16×	65.7%	85.9%
TBN	LS-BQNN [33]	Ternary/Binary	32×	>16×	62.0%	83.6%
BNN	CI-BCNN [44]	Binary (1-bit)	32×	64×	58.1%	80.0%

Dựa vào thông tin từ bảng 1-1 có thể đưa ra những đặc điểm để so sánh giữa mạng CNN, TNN và BNN như sau:

Bảng 1-2 So sánh đặc điểm giữa CNN, TNN và BNN

Đặc điểm	CNN	TNN	BNN
Biểu diễn thông tin	Số thực	Ba trạng thái (-1, 0, 1)	Hai trạng thái (-1,1)
Bộ nhớ	Cao	Trung bình	Thấp
Tốc độ	Chậm	Nhanh hơn CNN	Nhanh nhất
Độ chính xác	Cao	Trung bình	Thấp
Ứng dụng	Phổ biến trong xử lý ảnh, video	Thích hợp cho các hệ thống nhúng cần tiết kiệm năng lượng	Các thiết bị cần tối ưu bộ nhớ và tốc độ

Ternary Neural Network (TNN) đối mặt với một số vấn đề tồn động trong quá trình huấn luyện và triển khai. Thứ nhất, quá trình huấn luyện TNN có thể khó khăn do việc tối ưu hóa trong không gian trọng số rời rạc có thể phức tạp hơn so với mạng số thực. Thứ hai, việc sử dụng chỉ ba trạng thái (-1, 0, 1) để đại diện cho trọng số và kích hoạt có thể dẫn đến mất mát thông tin, giảm độ chính xác của mạng [4]. Cuối cùng, mặc dù TNN tiết kiệm bộ nhớ hơn so với mạng số thực, nhưng vẫn đòi hỏi một lượng tài nguyên tính toán vừa phải, đặc biệt khi triển khai trên các thiết bị có hạn về tài nguyên tính toán như các thiết bị nhúng.

Binary Neural Networks (BNN) mang lại một số ưu điểm so với TNN. Đầu tiên, BNN sử dụng chỉ hai trạng thái (-1 và 1) để đại diện cho trọng số và kích hoạt, giúp giảm bớt độ phức tạp của mạng so với TNN, nơi sử dụng ba trạng thái (-1, 0, 1). Sự đơn giản trong biểu diễn này giúp giảm bớt độ phức tạp của mạng, dễ dàng trong việc huấn luyện và triển khai . Thứ hai, BNN tiết kiệm bộ nhớ hơn so với TNN, vì chỉ cần lưu trữ hai trạng thái thay vì ba trạng thái. Điều này làm cho BNN trở nên hấp dẫn hơn đối với các ứng dụng có hạn về bộ nhớ, như trong các thiết bị di động và nhúng.

Mặc dù mô hình BNN có nhiều ưu điểm nhưng nó vẫn đang đối mặt với một số thách thức quan trọng cần được giải quyết trong tương lai. Thứ nhất, quá trình nhị phân hóa trọng số và kích hoạt dẫn đến mất mát thông tin nghiêm trọng. Điều này có thể ảnh hưởng đến độ chính xác và hiệu suất của mạng, đặc biệt là trong các tác vụ phức tạp đòi hỏi sự chi tiết cao và sự dao động trọng số làm giảm độ ổn định của BNN. Thứ hai, BNN có thể cần tài nguyên tính toán nhiều hơn trong quá trình huấn luyện so với TNN do số lượng trạng thái ít hơn, tạo ra một không gian trọng số rất hạn chế để tối ưu hóa. Việc huấn luyện trực tuyến có thể là giải pháp cho việc huấn luyện nhanh chóng và hiệu quả hơn, đặc biệt là cho các ứng dụng thời gian thực như bảo mật. Tuy nhiên, hiện chỉ có ít nghiên cứu về phương pháp này. Sự chuyển đổi dữ liệu và truy cập bộ nhớ vẫn tiêu tốn một phần đáng kể năng lượng trong quá trình thực thi của BNN và vấn đề thiết kế BNN không thể tổng quát cho tất cả các nhiệm vụ là những thách thức cần được nghiên cứu [5].

Bảng 1-3 So sánh một số phương pháp tiếp cận đối với mạng nơ-ron [6].

	Training			Deployment		
Method	**Inputs**	**Weights**	**Activations**	**Inputs**	**Weights**	**Activations**
BC [15], TC [9], TWN [8]	$\mathbb{R}$	$\{-1, 0, 1\}$	$\mathbb{R}$	$\mathbb{R}$	$\{-1, 0, 1\}$	$\mathbb{R}$
Binarized NN [6]	$\mathbb{R}$	$\{-1, 1\}$	$\{-1, 1\}$	$\mathbb{R}$	$\{-1, 1\}$	$\{-1, 1\}$
XNOR-Net[16]	$\mathbb{R}$	$\{-1, 1\}$	$\{-1, 1\}$ with $K, \alpha \in \mathbb{R}$	$\mathbb{R}$	$\{-1, 1\}$	$\{-1, 1\}$ with $K, \alpha \in \mathbb{R}$
EBP[17]	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$\{-1, 0, 1\}$	$\{-1, 1\}$
Bitwise NN [10]	$(-1, 1)$ $[0, 1]$	$(-1, 1)$ $(-1, 1)$	$(-1, 1)$ $(-1, 1)$	$\{-1, 1\}$ $\{0, 1\}$	$\{-1, 0, 1\}$ $\{-1, 0, 1\}$	$\{-1, 1\}$ $\{-1, 1\}$
TrueNorth [11]	$[0, 1]$	$[-1, 1]$	$[0, 1]$	$\{0, 1\}$	$\{-1, 0, 1\}$	$\{0, 1\}$
TNN (This Work)	$\{-1, 0, 1\}$	$\mathbb{R}$	$\{-1, 0, 1\}$	$\{-1, 0, 1\}$	$\{-1, 0, 1\}$	$\{-1, 0, 1\}$

Khi đem so sánh BNN với một số phương pháp khác trong bảng 1-3. Có thể thấy BNN là một lựa chọn lý tưởng so với các phương pháp khác khi mục tiêu là tối ưu hóa năng lượng và tài nguyên phần cứng. Một lý do chính là BNN sử dụng hoàn toàn các giá trị nhị phân cho cả trọng số và kích hoạt. Điều này giúp loại bỏ hoàn toàn các phép toán số thực, dẫn đến hiệu quả tối đa về năng lượng và giảm đáng kể tài nguyên phần cứng cần thiết. Trong khi đó, các phương pháp như Binary Connect (BC), Ternary Connect (TC) và Ternary Weight Networks (TWN) chỉ nhị phân hoặc tam phân hóa hóa trọng số, vẫn giữ kích hoạt ở dạng thực.

Phương pháp XNOR-Nets chuyển đổi các phép toán tích chập sang phép toán bitwise. Tuy nhiên, sau khi thực hiện phép toán bitwise để tính toán kết quả tích chập dựa trên các bit của ma trận đầu vào và kernel, kết quả này sẽ được nhân với các hệ số tỷ lệ (scaling factors) để thu được kết quả cuối cùng. Các hệ số tỷ lệ này thường là các số thực để điều chỉnh và cân bằng độ quan trọng của từng giá trị trong ma trận kết quả. Phương pháp Bitwise NN (Bitwise Neural Network) là một giải pháp hoàn toàn nhị phân, nhưng hiện tại không hỗ trợ CNN và cần hai bước huấn luyện, điều này làm tăng độ phức tạp. Trong khi đó, TrueNorth sử dụng chip thiết kế riêng cho mạng nơ-ron spiking với thuật toán back propagation cải tiến, nhưng cũng yêu cầu phần cứng chuyên dụng và chỉ mới mở rộng cho CNN.

Với mô hình thứ hai được đề cập chính trong đề tài. Mô hình SVM đòi hỏi nhiều thời gian và tài nguyên để chọn kernel. Việc tối ưu hóa siêu tham số còn ảnh hưởng đến huấn luyện, mà không có phương pháp nào đảm bảo hiệu quả để nén kích thước mô hình. SVM gặp khó khăn với chi phí tính toán tăng cao trong không gian đặc trưng cao [7].

Khả năng mở rộng và ứng dụng thực tế của BNN và SVM đều hạn chế, với BNN đòi hỏi thêm nghiên cứu để áp dụng trong các thiết bị nhúng và SVM cần cải thiện hiệu suất và kích thước mô hình. Cuối cùng, việc thiếu bộ tiêu chuẩn đánh giá chung cho cả hai mô hình làm khó khăn cho việc so sánh hiệu quả các phương pháp mới, gây trở ngại cho sự phát triển và ứng dụng thực tế. Những vấn đề này cho thấy cần có sự nghiên cứu và cải tiến liên tục để phát huy tối đa tiềm năng của cả hai mô hình trong các ứng dụng khác nhau.

Bên cạnh đó để tối ưu hóa xử lý mô hình trên thiết bị nhúng thì việc quản lý trạng thái thực thi tác vụ trên các thiết bị nhúng là một khía cạnh quan trọng nhằm đảm bảo hiệu suất và độ tin cậy của hệ thống. Thiết bị nhúng, với tài nguyên hạn chế về bộ nhớ và công suất xử lý, đòi hỏi các phương pháp quản lý hiệu quả để đảm bảo rằng các tác vụ quan trọng được thực hiện đúng thời gian và không gây ra tình trạng nghẽn hệ thống. Nhóm thực hiện đề tài đề xuất một framework là sự kết hợp giữa cơ chế Finite State Machine (FSM) và giao tiếp

Inter Process Communication (IPC) để hỗ trợ quản lý trạng thái tác vụ, đảm bảo rằng các tác vụ được xử lý kịp thời trong khi vẫn duy trì hiệu suất ổn định cho các tác vụ khác. Điều này đặc biệt quan trọng trong các ứng dụng như điều khiển công nghiệp, thiết bị y tế và các hệ thống tự động hóa, nơi mà độ tin cậy và phản ứng nhanh là yếu tố then chốt [8].

1.2 MỤC TIÊU ĐỀ TÀI

Mục tiêu chính của đề tài là phát triển các mô hình phân loại có kích thước nhỏ gọn và thời gian thực thi nhanh, đồng thời có khả năng ứng dụng trên các thiết bị nhúng. Đề tài còn bao gồm việc phát triển một ứng dụng sử dụng mô hình này và tích hợp với một framework quản lý trạng thái hệ thống.

Các mục tiêu cụ thể của đề tài bao gồm:

Nghiên cứu và phân tích các mô hình mạng nơ-ron và phương pháp thu nhỏ mô hình máy vector hỗ trợ (LSVM) để phù hợp với các thiết bị có tài nguyên hạn chế và yêu cầu tiết kiệm năng lượng, mà vẫn duy trì độ chính xác cao.

So sánh các tham số như độ chính xác, thời gian huấn luyện và bộ nhớ sử dụng để đánh giá hiệu suất của từng mô hình trong phân loại nhị phân. Từ các kết quả này, đề xuất mô hình phù hợp nhất cho từng ứng dụng cụ thể.

Phát triển giao diện cho việc triển khai các mô hình và tích hợp với framework quản lý trạng thái hệ thống.

Nhờ vào các nghiên cứu và thử nghiệm này, đề tài nhằm đóng góp vào việc tối ưu hóa hiệu suất và khả năng ứng dụng của các mô hình phân loại trên các thiết bị nhúng.

1.3 GIỚI HẠN ĐỀ TÀI

Đề tài tập trung nghiên cứu về mô hình BNN và LSVM. Xây dựng và triển khai mô hình trên phần cứng, từ đó đo đạt và đưa các nhận xét so sánh và hướng phát triển phù hợp.

Xây dựng cơ chế quản lý thái trên thiết bị nhúng, đảm bảo các trạng thái hoạt động đúng mà hiệu quả.

1.4 PHƯƠNG PHÁP NGHIÊN CỨU

Tiếp cận cơ sở lý thuyết của các mô hình, các công thức toán học liên quan. Xây dựng mô hình phân loại ảnh nhị phân và các phương pháp tối ưu mô hình. Thực hiện khảo sát các bài nghiên cứu khác để hiểu rõ hơn về các góc nhìn khác liên quan đến đề tài.

Thực hiện thử nghiệm: tiến hành huấn luyện và chạy thử nghiệm phân loại để đánh giá độ chính xác, thời gian thực thi, kích thước hình. Kết hợp mô hình với framework quản lý hệ thống

1.5 ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU

Đối tượng nghiên cứu: Nghiên cứu tập trung vào hai mô hình chính là Binary Neural Network (BNN) và Linear Support Vector Machine (LSVM). BNN sẽ được khảo sát về các thuật toán nhị phân hóa, bao gồm quá trình nhị phân hóa trọng số và kích hoạt, nhằm giảm thiểu mất mát thông tin và cải thiện hiệu suất. Đối với LSVM, nghiên cứu sẽ tập trung vào các phương pháp tối ưu hóa, đặc biệt là các phương pháp giảm kích thước mô hình và cải thiện hiệu suất tính toán. Đồng thời, nghiên cứu cũng bao gồm việc phát triển một framework quản lý trạng thái hệ thống kết hợp giữa FSM và IPC để đảm bảo hiệu suất và độ tin cậy của các thiết bị nhúng.

Phạm vi nghiên cứu: Phạm vi nghiên cứu bao gồm các thiết bị IoT nhỏ có tài nguyên hạn chế, như bộ nhớ, dung lượng lưu trữ và tốc độ xử lý, thường chạy bằng pin. Nghiên cứu sẽ áp dụng các kỹ thuật nén mô hình, lượng tử hóa và tỉa mô hình để giảm kích thước và yêu cầu tính toán của mô hình AI. Bên cạnh đó, nghiên cứu cũng sẽ đánh giá các biện pháp bảo mật và quyền riêng tư, xử lý dữ liệu nhạy cảm tại thiết bị biên. Hiệu suất và độ tin cậy của hệ thống sẽ được kiểm tra thông qua việc phát triển và triển khai framework quản lý trạng thái hệ thống trên các thiết bị nhúng, nhằm đảm bảo các tác vụ quan trọng được thực hiện đúng thời gian và duy trì hiệu suất ổn định.

1.6 BỐ CỤC QUYỂN BÁO CÁO

Nội dung chính của đề tài được trình bày với 5 chương:

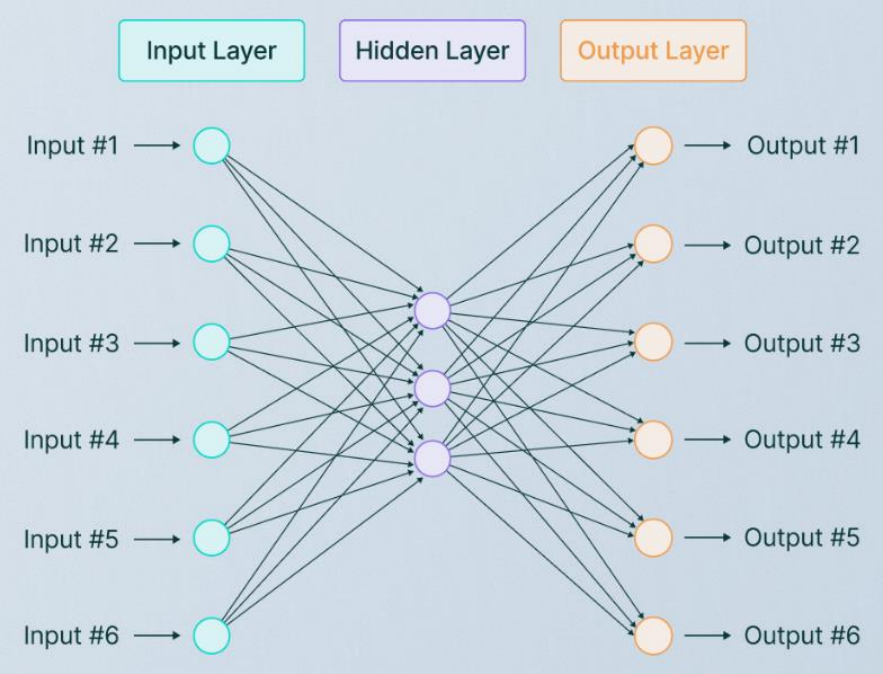
- Chương 1 GIỚI THIỆU: đặt vấn đề với đề tài, mục tiêu cần đạt được, giới hạn đề tài, phương pháp nghiên cứu, đối tượng và phạm vi nghiên cứu.
- Chương 2 CƠ SỞ LÝ THUYẾT: giới thiệu các mô hình, thuật toán và các phương pháp tối ưu hóa.
- Chương 3 XÂY DỰNG MÔ HÌNH VÀ THIẾT KẾ HỆ THỐNG: đưa ra mô hình, sơ đồ khối cho các mô hình, mã giả. Biểu diễn các thành phần của cơ chế quản lý hệ thống.
- Chương 4 KẾT QUẢ: trình bày kết quả đạt được từ các mô hình và cơ chế quản lý.
- Chương 5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN: rút ra các kết luận, hạn chế và hướng phát triển của mô hình.

CHƯƠNG 2

CƠ SỞ LÝ THUYẾT

2.1 MÔ HÌNH MẠNG NƠ-RON

Mạng nơ-ron là thành phần cốt lõi của học sâu và được biết đến với khả năng mô phỏng hành vi của bộ não con người để giải quyết các vấn đề phức tạp dựa trên dữ liệu. Dữ liệu đầu vào được xử lý qua các lớp của nơ-ron nhân tạo xếp chồng lên nhau để tạo ra đầu ra mong muốn. Từ nhận diện giọng nói và nhận diện người cho đến chăm sóc sức khỏe và tiếp thị, mạng nơ-ron đã được sử dụng trong nhiều lĩnh vực khác nhau.



Hình 2-1 Mô hình mạng nơ-ron

Lớp đầu vào nhận dữ liệu từ các nguồn bên ngoài và chuyển thông tin vào mạng nơ-ron mà không thực hiện tính toán. Các lớp ẩn trung gian thực hiện các phép tính và trích xuất đặc trưng từ dữ liệu, có thể bao gồm nhiều lớp liên kết để

tìm kiếm các đặc trưng ẩn khác nhau. Lớp đầu ra nhận thông tin từ các lớp ẩn trước đó và đưa ra dự đoán cuối cùng dựa trên học tập của mô hình, với cấu trúc cụ thể phụ thuộc vào loại bài toán [9].

2.2 MÔ HÌNH MẠNG CNN

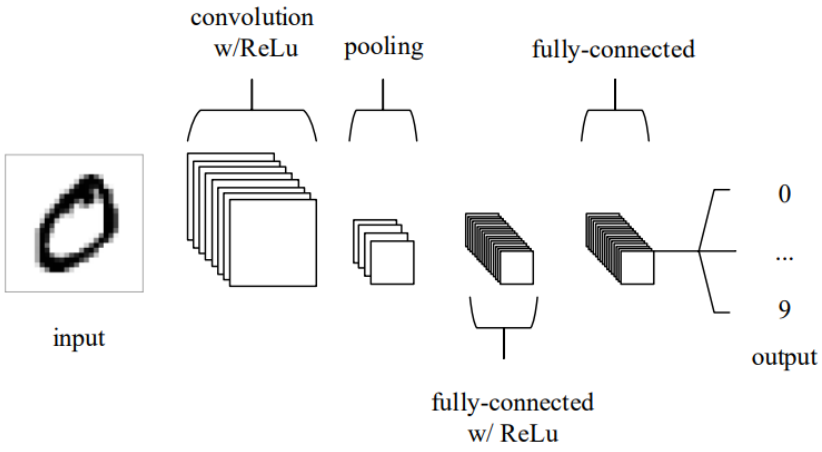
2.2.1 Sơ lược về mạng CNN

Mạng Nơ-ron tích chập (Convolutional Neural Network - CNN) tương tự như các mạng rơ-ron truyền thống ở chỗ chúng cũng bao gồm các nơ-ron tự tối ưu hóa thông qua việc học. Mỗi nơ-ron sẽ nhận một đầu vào và thực hiện một phép toán (chẳng hạn như tích vô hướng theo sau bởi một hàm phi tuyến). Từ các vectơ hình ảnh thô đầu vào đến đầu ra cuối cùng của trọng số lớp, toàn bộ mạng lưới sẽ vẫn thể hiện một hàm trọng số nhận thức duy nhất. Lớp cuối cùng sẽ chứa các hàm mất mát liên quan đến các lớp.

Điểm khác biệt đáng chú ý duy nhất giữa CNN và nơ-ron truyền thống là CNN chủ yếu được sử dụng trong lĩnh vực nhận dạng mẫu trong hình ảnh. Điều này cho phép chúng ta mã hóa các đặc điểm cụ thể của hình ảnh vào kiến trúc, làm cho mạng lưới phù hợp hơn cho các nhiệm vụ tập trung vào hình ảnh - đồng thời giảm thêm số lượng tham số cần thiết để thiết lập mô hình.

2.2.2 Kiến trúc CNN

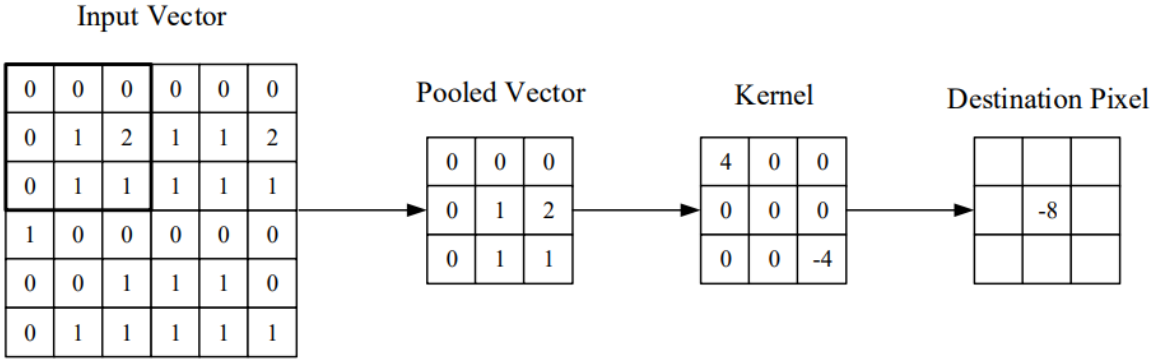
CNN bao gồm ba lớp: lớp convolutional, lớp pooling và lớp fully-connected. Các lớp này xếp chồng lên nhau sẽ tạo thành kiến trúc CNN [10].



Hình 2-2 Kiến trúc mạng CNN cơ bản [11].

Hình 2-2 là kiến trúc CNN đơn giản hóa để phân loại MNIST. Giống như các mạng nơ-ron khác, lớp đầu vào sẽ chứa các giá trị pixel của hình ảnh.Lớp convolutional áp dụng các bộ lọc cho hình ảnh đầu vào để trích xuất các đặc điểm, lớp pooling sau đó sẽ thực hiện giảm mẫu theo chiều không gian của đầu vào đã cho, giảm thêm số lượng tham số trong kích hoạt đó và lớp fully-connected sẽ tạo ra các trọng số từ các hàm kích hoạt và sử dụng để phân loại. Hàm ReLu có thể được sử dụng giữa các lớp này để cải thiện hiệu suất.

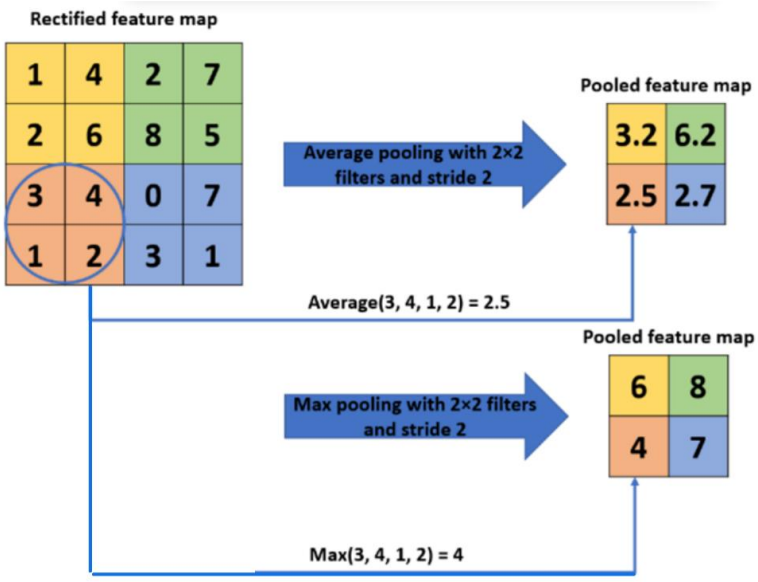
Lớp convolutional đóng vai trò quan trọng trong cách CNN hoạt động. Các tham số của lớp này sử dụng các kernel có thể học được. Những kernel này thường có kích thước nhỏ theo chiều không gian và trải dọc theo chiều sâu của đầu vào. Khi dữ liệu qua lớp convolutional, lớp này sẽ tích chập mỗi bộ lọc trong chiều không gian để tạo ra bản đồ hai chiều.



Hình 2-3 Lớp convolutional của mô hình CNN [11].

Theo hình 2-3, khi chúng ta di chuyển qua đầu vào, tích vô hướng được tính cho mỗi giá trị trong kernel đó. Từ đó, mạng sẽ học các kernel "vùng nhiệt" khi chúng phát hiện một đặc trưng cụ thể tại một vị trí nhất định gọi là hàm kích hàm kích hoạt. Cuối cùng, mỗi kernel sẽ có một bản đồ kích hoạt tương ứng, sẽ được xếp chồng theo chiều sâu để tạo thành số lượng đầu ra đầy đủ của lớp convolutional.

Lớp pooling sẽ có nhiệm vụ giảm dần kích thước của bản đồ, giảm số lượng tham số và độ phức tạp tính toán của mô hình nhưng vẫn giữ lại thông tin quan trọng. Lớp pooling hoạt động trên mỗi bản đồ kích hoạt trong đầu vào và thu nhỏ kích thước của nó bằng cách sử dụng hàm max pooling. Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình. Tổng tất cả các phần tử trong map gọi là sum pooling. Trong hầu hết các CNN, các lớp này xuất hiện dưới dạng các lớp max-pooling với kernel có kích thước 2 × 2, được áp dụng với bước nhảy (stride) dọc theo các chiều không gian của đầu vào. Điều này làm thu nhỏ bản đồ kích hoạt xuống còn 25% kích thước ban đầu - trong khi vẫn duy trì chiều sâu của khối lượng như kích thước chuẩn.

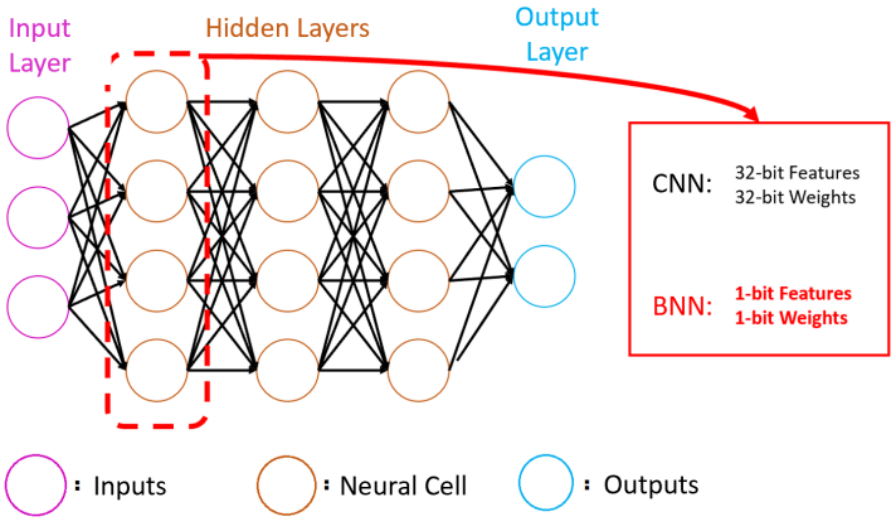


Hình 2-4 Hình tượng hóa lớp Pooling với average pooling và max pooling [12].

Lớp fully-connected nhận dữ liệu đầu vào đã được làm phẳng, mỗi đầu vào đó được kết nối trực tiếp với tất cả các nơ-ron để tối ưu hóa mục tiêu của mô hình.

2.3 MÔ HÌNH MẠNG BNN

Mạng nơ-ron nhị phân (Binary Neural Network - BNN) là một loại mạng nơ-ron trong đó các hàm kích hoạt và trọng số là các giá trị 1-bit trong tất cả các lớp ẩn. Nói một cách đơn giản, BNN là một phiên bản rút gọn của CNN. Bởi vì BNN và CNN có cấu trúc giống nhau, chỉ khác giá trị kích hoạt và trọng số với độ chính xác khác nhau.

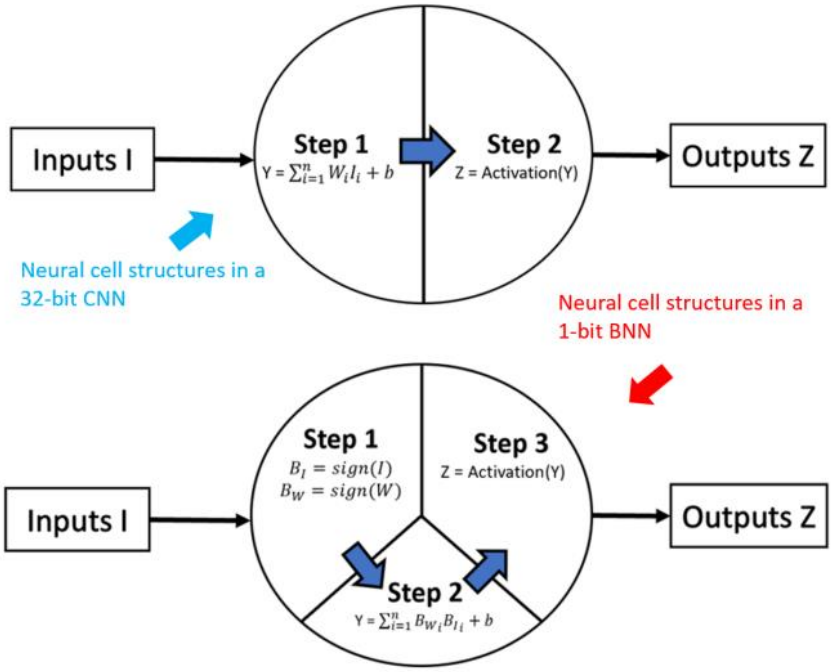


Hình 2-5 Đặc điểm mô hình BNN so với CNN

BNN sử dụng các kỹ thuật để nén các giá trị hàm kích hoạt và trọng số từ 32-bit thành các giá trị 1-bit gọi. Quá trình nén 32-bit thành 1-bit được gọi là quá trình nhị phân hóa với mục đích vừa để tiết kiệm bộ nhớ vừa để giảm các phép toán ma trận. BNN có thể tiết kiệm bộ nhớ lên đến 32 lần và thực hiện phép toán nhanh gấp 58 lần so với CNN 32-bit.

2.3.1 Lan truyền xuôi (Forward Propagation)

Mỗi nơ-ron là một phép toán cơ bản trong đường truyền xuôi của mạng nơ-ron. Khác với CNN 32-bit, các nơ-ron của BNN sẽ thêm bước nhị phân hóa với các hàm kích hoạt I và trọng số W trước khi thực hiện phép tích chập, với mục đích biểu diễn số thực bằng 1-bit.



Hình 2-6 So sánh hàm kích hoạt của mô hình BNN với CNN

Hàm nhị phân hóa:

$$Sign(x) = \begin{cases} +1, if\ x \geq 0, \\ -1, \quad \neq \end{cases}$$

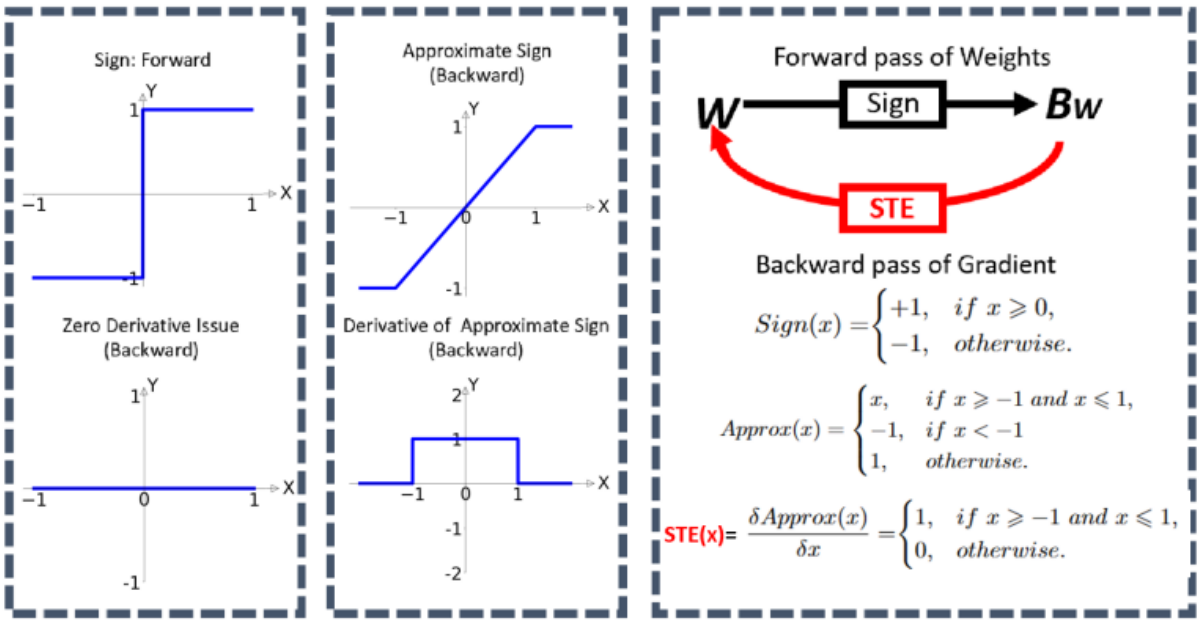
Sau nhị phân hóa, hàm kích hoạt I và trọng số W được biểu diễn:

$$I \approx sign(I) = B_i$$

$$W \approx signh(W) = B_W$$

2.3.2 Lan truyền ngược (Backward Propagation)

Do kết quả đạo hàm của hàm nhị phân hóa (sign) bằng 0, các trọng số nhị phân không thể được học bằng phương pháp gradient descent truyền thống dựa trên thuật toán lan truyền ngược. Để giải quyết vấn đề này, BNN áp dụng kỹ thuật gọi là ước lượng trực tiếp (Straight-Through Estimator - STE) để học các trọng số nhị phân trong quá trình lan truyền ngược.



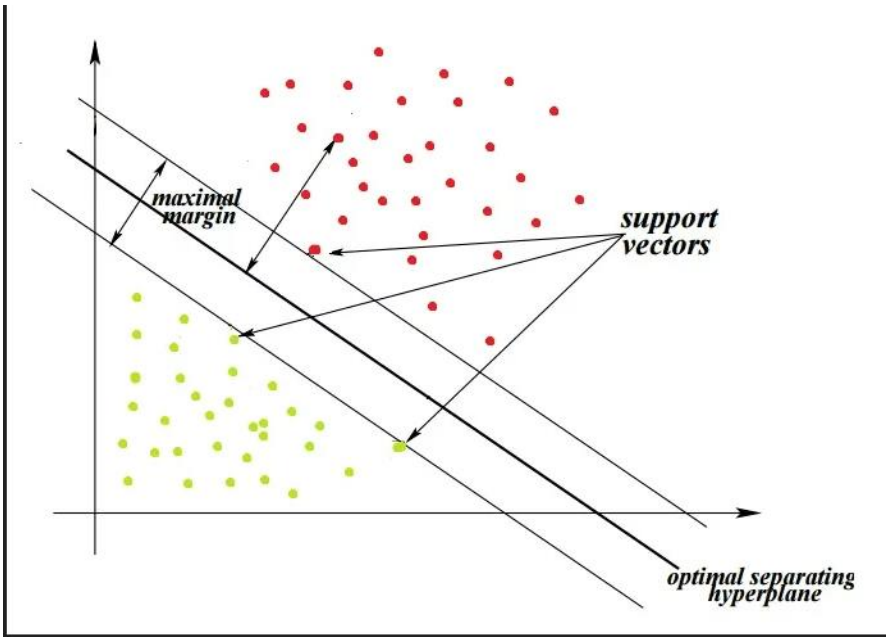
Hình 2-7 Xử lý tính toán trong lan truyền ngược

Hình 2-7 giải thích quá trình học các trọng số nhị phân trong BNN. Trong các bước huấn luyện BNN, trọng số thực của mỗi lớp được giữ lại và cập nhật bằng cách sử dụng STE. Sau khi huấn luyện, các trọng số nhị phân được lưu lại và trọng số thực bị loại bỏ. Ngoài ra, BNN bao gồm hai bước để huấn luyện mô hình. Bước đầu tiên là huấn luyện một số tham số mạng nén trong các mạng giá trị thực với việc nén trọng số. Sau đó, khởi tạo các tham số giá trị thực cho mạng nơ-ron bitwise mục tiêu và áp dụng chiến lược huấn luyện tương tự STE [13].

2.4 MÔ HÌNH SVM

2.4.1 Sơ lược về SVM

Support Vector Machine (SVM) là một thuật toán máy học có giám sát được sử dụng khá phổ biến trước khi mạng nơ ron ra đời. Đây là một thuận toán khá hiệu quả đối với các bài toán phân loại nhị phân. SVM hoạt động bằng cách tìm ra một siêu phẳng (hyperplane) để phân tách dữ liệu thuộc các lớp khác nhau trong không gian N chiều tương ứng với N đặc trưng.



Hình 2-8 Hình tượng hóa SVM

2.4.2 Các khái niệm chính

Siêu phẳng (hyperplane) là một không gian con của không gian nhiều chiều mà thuật toán sử dụng để phân tách các lớp. Đối với không gian hai chiều thì siêu mặt là một đường thẳng, trong không gian ba chiều thì nó trở thành một mặt phẳng.

Lề (Margin) là khoảng cách giữa siêu phẳng đến các điểm dữ liệu gần nhất của mỗi lớp. Thuật toán sẽ tạo ra các siêu phẳng sao cho khoảng cách đến các điểm dữ liệu gần nhất thuộc hai lớp khác nhau là lớn nhất. Điều này giúp cải thiện khả năng tổng quát của mô hình.

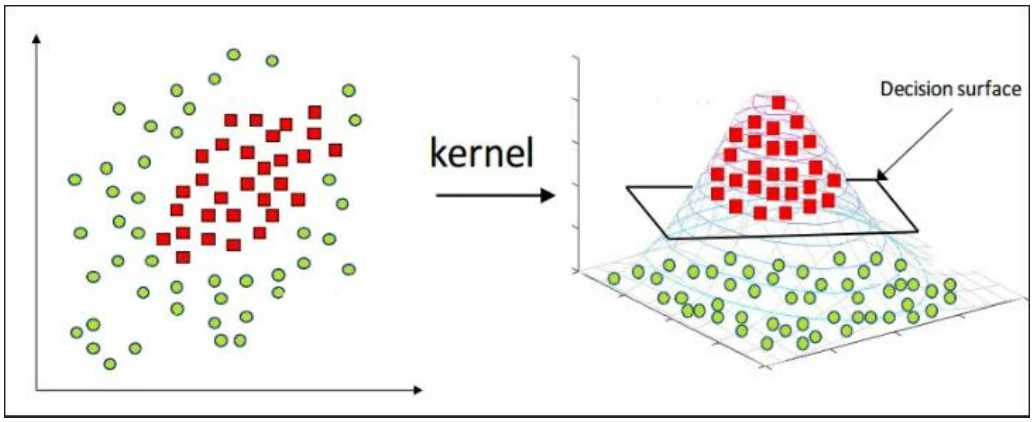
Support vector là các điểm dữ liệu gần nhất với siêu phẳng. Chúng có vai trò quan trọng trong việc xác định vị trí của siêu phẳng.

SVM có 2 kiểu phân tách:

+ Phân tách tuyến tính: Khi dữ liệu có thể phân tách bằng một siêu phẳng tuyến tính, SVM sẽ tìm kiếm siêu phẳng này và thường được áp dụng cho các tập dữ liệu đơn giản.

+ Phân tách phi tuyến: Khi dữ liệu không thể phân tách tuyến tính trong không gian đầu vào, SVM sẽ sử dụng kỹ thuật kernel trick để ánh xạ dữ liệu vào

một không gian nhiều chiều hơn để tìm siêu phẳng tuyến tính để phân tách dữ liệu.



Hình 2-9 Kernel của SVM phi tuyến

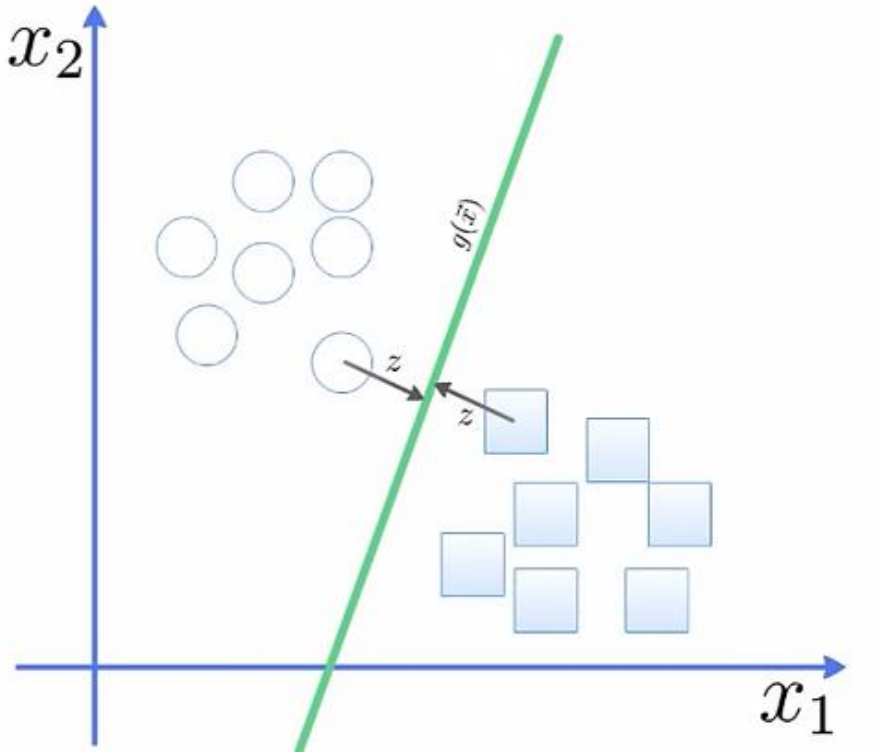
Ưu điểm:

+ Thuật toán hoạt động hiệu quả với không gian nhiều chiều.
+ Tiết kiệm bộ nhớ do chỉ lưu trữ những điểm dữ liệu (support vectors) quan trọng.
+ Linh hoạt với kernel trick.

Nhược điểm:

+ Không hiệu quả với các tập dữ liệu lớn vì khả năng tính toán phức tạp.
+ Hiệu suất mô hình có thể giảm nếu dữ liệu bị nhiễu.
+ Việc chọn kernel phù hợp là một thách thức và cần nhiều thử nghiệm [14].

2.4.3 Mô hình thuật toán SVM

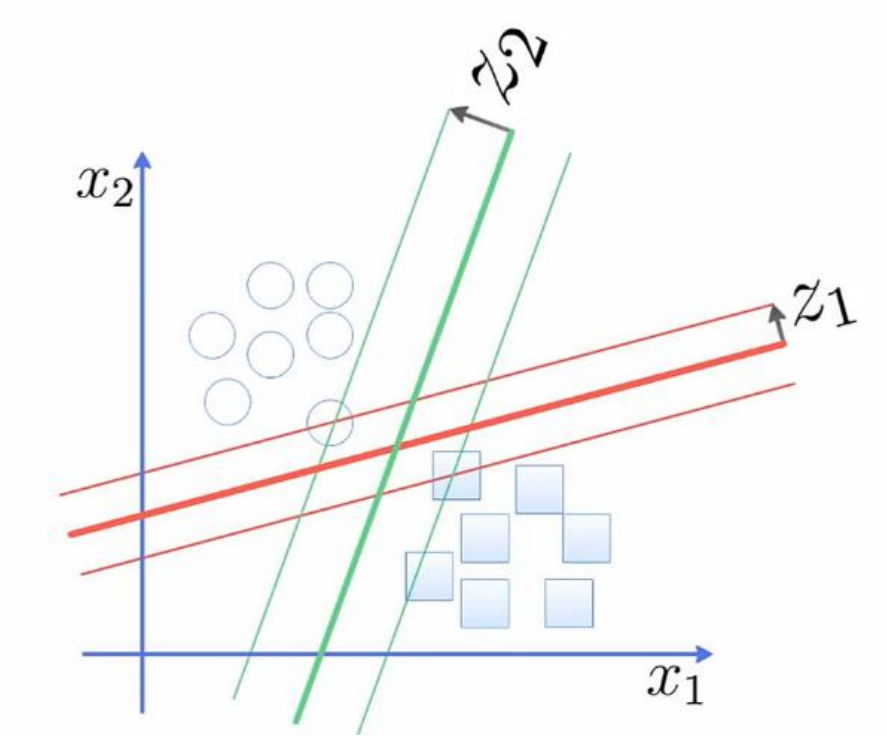


Hình 2-10 Biểu diễn siêu phẳng của SVM trong không gian

Mục tiêu là tìm ra mặt siêu phẳng để phân loại tất cả vector huấn luyện trong 2 lớp với công thức:

$$g(\vec{x}) = \vec{\omega}^T \vec{x} + \omega_0$$

$$z = \frac{|g(\vec{x})|}{||\vec{\omega}||} = \frac{1}{||\vec{\omega}||}$$ [15]

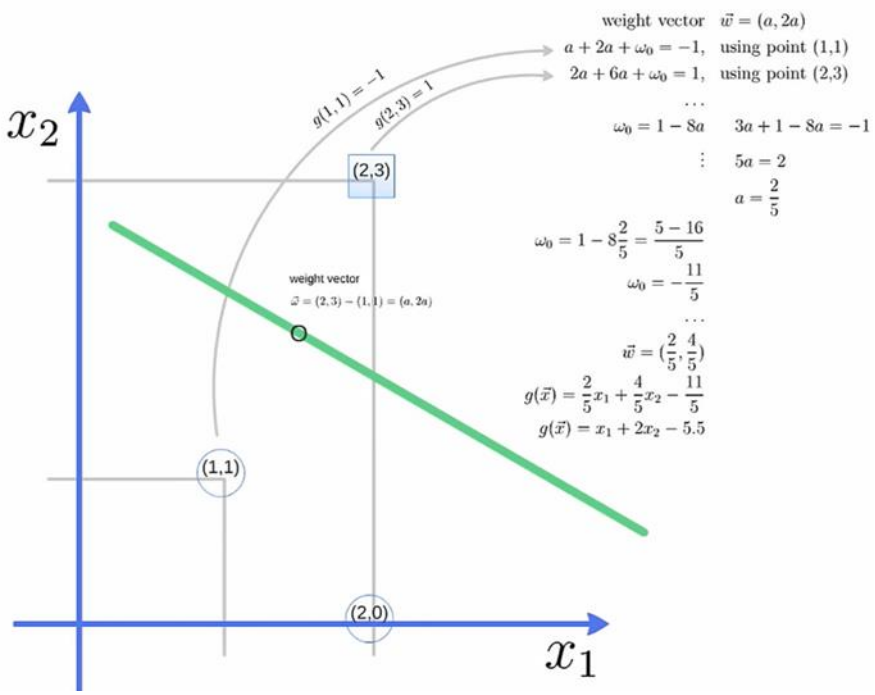


Hình 2-11 Biểu diễn lề của siêu mặt trong không gian

Để xét xem siêu phẳng màu xanh hay màu đỏ tốt hơn thì ta cần đề cập đến lề (margin) Z_1 và Z_2, ta có thể thấy $Z_1 > Z_2$ nên siêu phẳng màu xanh sẽ tối ưu hơn.

$$g(\vec{x}) \geq 1, \forall \vec{x} \in \text{nhóm 1}$$

$$g(\vec{x}) \leq -1, \forall \vec{x} \in \text{nhóm 2}$$



Hình 2-12 Tính toán khoảng cách điểm đến siêu phẳng

Trong không gian F nhiều chiều hơn sẽ dẫn đến việc tính toán tốn nhiều bộ nhớ và thời gian. Để có thể xử lý tính toán việc này dễ hơn, chúng ta sử dụng các hàm kernel function:

- Polytniminal: $\mathrm{K}(\boldsymbol{\omega}, \mathbf{x}) = (\gamma\vec{\omega}\mathrm{x} + \mathrm{b})^{\mathrm{N}}$

- Gaussian RBF : $K(x, y) = e^{-\gamma\|x-y\|^2}$

- Sigmoid: $K(x_i, x_j) = \tanh(\alpha {x_i}^T x_i + b)$ [16]

2.5 THUẬT TOÁN K-MEANS

2.5.1 Lý thuyết

K-Means là một thuật toán phân cụm dùng để giải quyết bài toán phân cụm. Ý tưởng của K-Means là phân chia một bộ dữ liệu thành các cụm khác nhau với số lượng cụm được cho trước gọi là k. Các điểm dữ liệu trong một cụm phải có cùng một số đặc trưng nhất định tức là giữa các điểm trong cụm phải liên quan đến nhau. Thuật toán phân cụm K-Means thường được sử dụng trong các ứng dụng phân đoạn khách hàng, thống kê dữ liệu,....

Hình 2-13 Biểu diễn tâm cụm với K-Means [17].

2.5.2 Thuật toán phân cụm

1: Khởi tạo k điểm dữ liệu trong bộ dữ liệu tương ứng với k cụm mong muốn.

2: Tương ứng với k điểm dữ liệu chọn ra k điểm trung tâm để làm tâm cụm.

3: Gán các điểm dữ liệu còn lại vào k điểm trung tâm, dựa vào khoảng cách giữa điểm đó và tâm cụm.

4: Tính tâm cụm mới ở mỗi cụm bằng cách lấy trung bình cộng của tất cả các điểm dữ liệu trong cụm đó. Công thức tính toán trung bình cộng cho mỗi tọa độ là:

$$\text{New center} = \frac{1}{N}\sum_{i=1}^{N} point_i$$

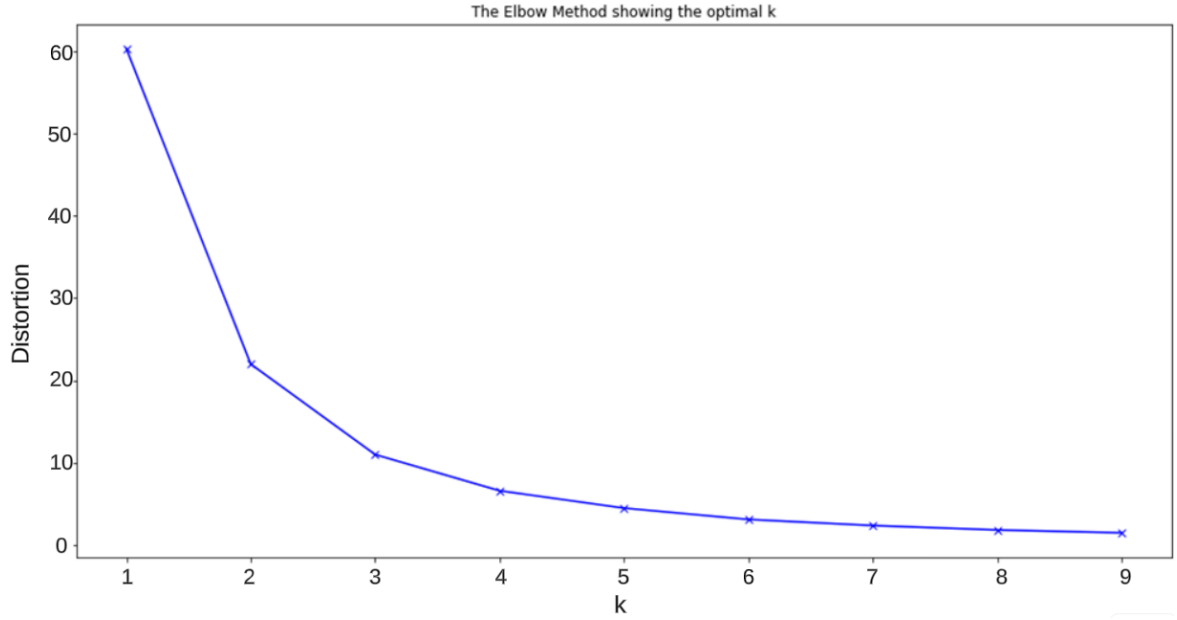
Trong đó, N là số lượng điểm trong cụm và $point_i$là điểm thứ i trong cụm.

5: Lặp lại từ bước 3 cho đến khi tâm không đổi.

Để xác định trước số cụm ta có các phương pháp sau:

Phương pháp elbow: là một cách giúp chúng ta lựa chọn được số cụm phù hợp dựa vào biểu đồ sự suy giảm của hàm biến dạng và lựa chọn điểm khuỷu tay (elbow point). Ví dụ ta có k cụm, mỗi điểm dữ liệu bất kì điều thuộc vào một

cụm, ta sẽ tính trung bình cộng bình phương khoảng cách giữa tâm cụm đến các cụm còn lại gọi là Distortion.



Hình 2-14 Biểu đồ distortion tương ứng với số cụm trong thuật toán elbow

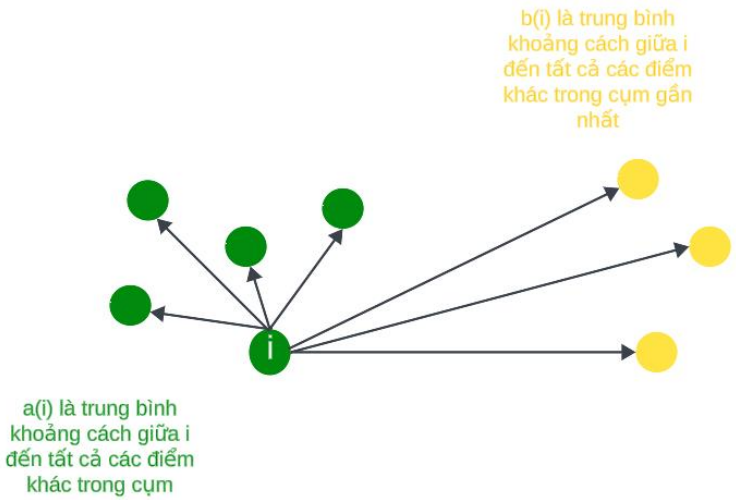
Để xác định số cụm tối ưu, chúng ta phải chọn giá trị k ở "khuỷu tay", tức là điểm mà sau đó độ méo/quán tính bắt đầu giảm theo kiểu tuyến tính. Do đó, đối với dữ liệu đã cho, nhóm thực hiện đề tài kết luận rằng số cụm tối ưu cho dữ liệu là 4 [18].

Phương pháp Silhouett: là một kỹ thuật để đánh giá chất lượng của việc phân cụm trong thuật toán K-Means và các thuật toán phân cụm khác. Silhouette score đo lường mức độ tương đồng của một điểm dữ liệu với cụm của nó so với các cụm khác. Điểm số này giúp xác định số lượng cụm tối ưu và đánh giá hiệu quả của việc phân cụm.

$$\text{Sihouette score} = \frac{b-1}{\max(a,b)} \quad [19]$$

Trong đó :

+ a là trung bình cộng khoảng cách của một điểm hiện tại đến tất cả các điểm trong cụm.
+ b là trung bình cộng khoảng cách của một điểm đến tất cả các điểm của cụm gần nhất.



Hình 2-15 Biểu diễn tính Sihouette trong không gian

Giá trị Sihouette score nằm trong khoảng từ -1 đến 1:

+ Sihouette score càng gần 1 thì điểm dữ liệu được phân cụm rất tốt.
+ Sihouette score gần 0 thì điểm dữ liệu nằm gần biên giới giữa hai cụm.
+ Sihouette score nhỏ hơn 0 thì điểm dữ liệu đang được phân cụm sai [20].

Bảng 2-1 So sánh phương pháp Elbow và phương pháp Silhouette

Đặc điểm	Phương pháp Elbow	Phương pháp Silhouette
Mục tiêu	Xác định điểm khuỷu tay của SSE/Distortion	Đánh giá mức độ phân tách giữa các cụm
Cách tiếp cận	Quan sát đồ thị SSE theo k	Tính toán Silhouette score cho từng điểm
Độ phức tạp tính toán	Thấp	Cao
Đánh giá chất lượng	Dựa trên Distortion	Dựa trên khoảng cách giữa các cụm
Ưu điểm	Dễ hiểu, dễ triển khai	Phản ánh tốt chất lượng phân cụm
Hạn chế	Điểm khuỷu tay không luôn rõ ràng	Tính toán phức tạp, tốn thời gian
Phù hợp với dữ liệu lớn	Tốt	Kém hơn

2.6 GIẢI THUẬT LƯU MÔ HÌNH NHỊ PHÂN

2.6.1 BitArray

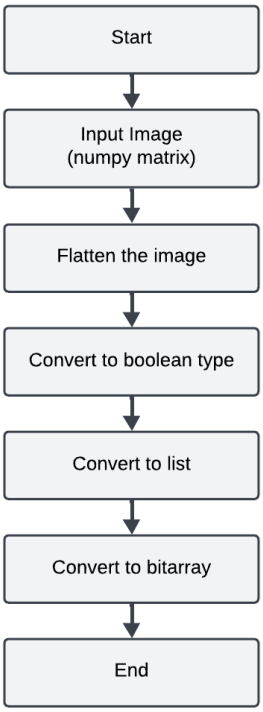
BitArray là một collection module giúp quản lý, lưu trữ một danh sách các bit (0 hoặc 1), được biểu diễn như kiểu Boolean. Trong đó true biểu thị cho bit 1 và false biểu thị cho bit 0.

Các tính năng chính:

+ Tiết kiệm bộ nhớ: BitArray giúp tiết kiệm bộ nhớ hơn rất nhiều. Mặc dù kiểu Boolean chỉ lưu 2 giá trị true và false nhưng lại tốn đến 1 bytes cho mỗi biến kiểu Boolean. Trong khi đó mỗi phần tử trong BitArray tốn đúng 1 bit để lưu trữ.
+ Hiệu suất cao: Thư viện này được viết bằng C, giúp tăng cường hiệu suất khi thực hiện các thao tác bitwise.

+ Dễ dàng sử dụng: Cung cấp các phương thức tiện lợi để thao tác với các bit, chẳng hạn như thêm, xóa, đảo ngược và truy cập các bit.

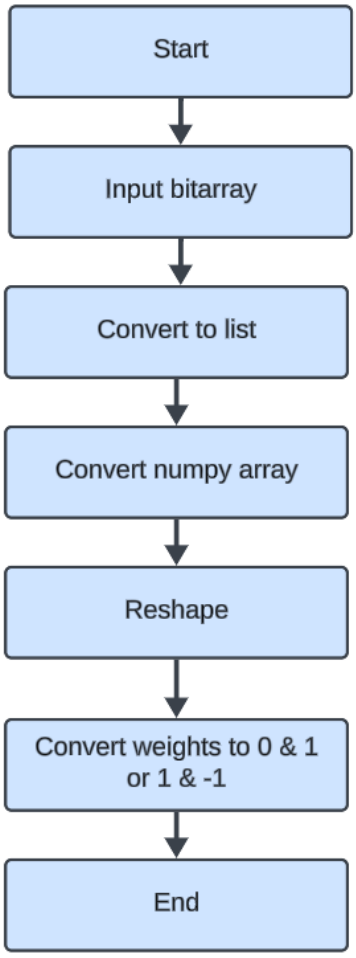
2.6.2 Mã hóa



Hình 2-16 Quá trình mã hóa BitArray

Khi mô hình cần lưu trữ ma trận dung lượng thấp. Hãy mã hóa chúng như sau. Ma trận numy đầu phải được duỗi thẳng thành mảng sau đó chúng ta chuyển đổi chúng thành kiểu Boolean. Từ đây chuyển tiếp thành kiểu list và sử dụng BitArray để có được ma trận 1-bit. Kết quả chúng ta nhận được một mảng 1-bit thay vì một ma trận số nguyên 32-bit.

2.6.3 Giải mã



Hình 2-17 Quá trình giải mã BitArray

Để chuyển đổi mảng BitArray. Đầu tiên, phải chuyển đổi nó về dạng list sau đó chuyển tiếp về mảng numpy. Tại đây, chúng ta sẽ nhập kích thước ma trận chúng ta muốn khôi phục. Sau đó chúng ta có thể chuyển đổi chúng về dạng 0 và 1 hoặc 1 và -1 tùy thuộc vào nhu cầu sử dụng [21].

2.7 PHƯƠNG PHÁP ĐO TÀI NGUYÊN SỬ DỤNG CỦA MÔ HÌNH

2.7.1 Memory profiler

Thư viện memory_profiler là một công cụ phổ biến trong Python để theo dõi việc sử dụng bộ nhớ của các chương trình Python. Nó giúp các nhà phát triển hiểu rõ hơn về mức độ tiêu thụ bộ nhớ của mã nguồn, từ đó tối ưu hóa và cải thiện hiệu năng của các ứng dụng. Dưới đây là một số điểm chính về memory_profiler:

Tính năng chính:

+ Theo dõi bộ nhớ: memory_profiler cho phép theo dõi mức sử dụng bộ nhớ của từng dòng mã Python trong thời gian thực.
+ Dễ sử dụng: Cung cấp cách tiếp cận đơn giản để sử dụng, thông qua việc thêm các decorator vào các hàm cần theo dõi.
+ Tích hợp với IPython/Jupyter: Có khả năng tích hợp tốt với IPython và Jupyter Notebook, giúp theo dõi bộ nhớ trong môi trường tương tác.
+ Báo cáo chi tiết: Tạo ra các báo cáo chi tiết về việc sử dụng bộ nhớ, giúp xác định các đoạn mã ngốn nhiều bộ nhớ nhất.

2.7.2 Cách thực hiện

Bước 1: Cài đặt thư viện trên terminal:

Hình 2-18 Lệnh cài thư viện memory profiler

Cài đặt thư viện memory_profiler sử dụng command "pip install memory_profiler".

Bước 2: Sử dụng với decorator:

Tiến hành khai báo thư viện memory_profiler vào chương trình. Sau đó, đặt lệnh "@profile" ở phía trên hàm cần đo.

```python
from memory_profiler import profile

@profile
def make_predictions(test_data):
    results = []
    for image, labels in zip(test_data['image'], test_data['label']):
        # Reshape image to fit model input
        image_array = image.reshape(1, 28, 28, 1)

        # Predict the label
        prediction = model.predict(image_array)
        label = np.argmax(prediction)

        # Store the result
        results.append((labels, label))

    return results
```

Khai báo thư viện memory_profilerprofiler

Đặt Decorator ở phía trên hàm cần đo

Hình 2-19 Sử dụng thư viện memory profiler trong chương trình

Sau đó, mở terminal và sử dụng lệnh sau để tiến hành đo:

```
jetson@nano:~/Downloads$ cd /home/jetson/Downloads/SourceTest
jetson@nano:~/Downloads/SourceTest$ python3 -m memory_profiler Test_CNN.py
```

Hình 2-20 Lệnh đo tài nguyên sử dụng của hàm

Hoặc lệnh sau:

```
jetson@nano:~/Downloads/SourceTest$ mprof run Test_CNN.py
mprof: Sampling memory every 0.1s
running new process
running as a Python program...
```

Hình 2-21 Lệnh thay thế đo tài nguyên sử dụng của hàm

Bước 3: Xem kết quả:

```
jetson@nano: ~/Downloads/SourceTest
2024-06-10 18:36:06.303214: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic
 library libcudnn.so.8
Filename: Test_CNN.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    34   1510.9 MiB   1510.9 MiB           1   @profile
    35                                         def make_predictions(test_data):
    36   1510.9 MiB      0.0 MiB           1       results = []
    37   2092.6 MiB -45321.2 MiB         271       for image, labels in zip(test_data['image'], test_data['label']):
    38                                                 # Reshape image to fit model input
    39   2092.6 MiB -45088.6 MiB         270           image_array = image.reshape(1, 28, 28, 1)
    40
    41                                                 # Predict the label
    42   2092.4 MiB -44739.1 MiB         270           prediction = model.predict(image_array)
    43   2092.6 MiB -45261.9 MiB         270           label = np.argmax(prediction)
    44
    45                                                 # Store the result
    46   2092.6 MiB -45321.1 MiB         270           results.append((labels, label))
    47
    48   1859.7 MiB   -232.9 MiB           1       return results
```

Hình 2-22 Kết quả đo đạt tài nguyên sử dụng

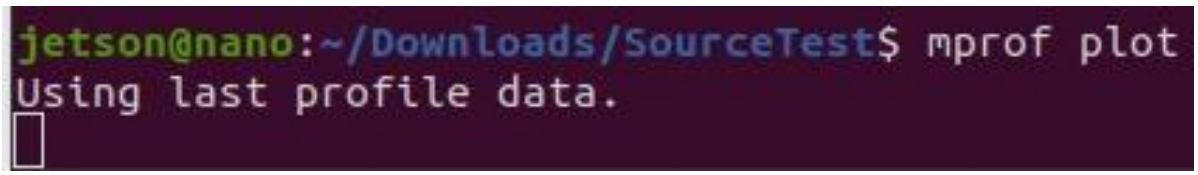
Giải thích các thông số đo được:

+ Line : số dòng trong file cần đo Memory Usage.
+ Mem Usage : bộ nhớ sử dụng tại thời điểm đó.
+ Increment : độ chênh lệch bộ nhớ sử dụng giữa thời điểm hiện tại và trước đó.
+ Occurrences : số lần thực thi.
+ Line Content: nội dung dòng code.

* Lưu ý khi đo bằng lệnh "mprof run", sau khi chạy xong chúng ta nhận được một file ".dat". Khi đọc file này cũng thu được kết quả tương tự.

Hình 2-23 file lưu dữ liệu tài nguyên sử dụng

Ngoài ra sau khi đo đạt xong, có thể sử dụng lệnh "mprof plot" để vẽ biểu đồ bộ nhớ sử dụng theo thời gian [22].



Hình 2-24 Lệnh vẽ biểu đồ tài nguyên sử dụng

Cuối cùng, chúng ta sẽ xem được biểu đồ như hình 4-7.

2.8 PHƯƠNG PHÁP ĐO THỜI GIAN THỰC THI

2.8.1 Time

Thư viện time là một trong những thư viện chuẩn của Python, cung cấp các chức năng để làm việc với thời gian và ngày tháng. Thư viện này cung cấp các tác vụ như lấy thời gian hiện tại, chuyển đổi giữa các định dạng thời gian, tính toán khoảng thời gian hoặc cho hệ thống ngủ trong một khoảng thời gian nhất định.

Tính năng chính:

+ Lấy thời gian hiện tại: time.time() trả về thời gian hiện tại tính bằng giây từ Unix epoch (1970-01-01 00:00:00 UTC).
+ Trì hoãn thực thi chương trình: time.sleep(secs) tạm dừng chương trình trong một khoảng thời gian nhất định (đơn vị: giây).
+ Chuyển đổi thời gian: time.ctime(secs) chuyển đổi thời gian từ giây thành chuỗi dễ đọc, time.gmtime(secs) và time.localtime(secs) chuyển đổi thời gian thành cấu trúc thời gian theo múi giờ UTC hoặc địa phương.
+ Định dạng thời gian: time.strftime(format, t) chuyển đổi cấu trúc thời gian thành chuỗi theo định dạng chỉ định.
+ Phân tích chuỗi thời gian: time.strptime(string, format) phân tích chuỗi thời gian thành cấu trúc thời gian dựa trên định dạng chỉ định.
+ Đo thời gian thực thi: time.perf_counter() trả về giá trị thời gian chính xác nhất của hệ thống, dùng để đo khoảng thời gian thực thi của các đoạn mã.

2.8.2 Cách thực hiện

Bước 1: Lấy thời gian bắt đầu:

```
1   import time
2   # Bắt đầu đo thời gian
3   start_time = time.time()
```

Hình 2-25 Đặt mốc thời gian bắt đầu đo

Trước khi hàm thực thi nhóm sẽ lưu thời gian hiện tại bằng cách gọi time.time(). Giá trị này được lưu trong một biến, thường gọi là start_time.

Bước 2: Xác định hàm cần đo thời gian thực thi và đặt mốc lấy thời gian sau khi hàm thực thi xong:

```
import time
# Bắt đầu đo thời gian
start_time = time.time()
# Gọi hàm tính toán độ chính xác
accuracy = calculate_accuracy(test_data, Wh, bh, Wo, bo)
# Kết thúc đo thời gian
end_time = time.time()
```

Hình 2-26 Gọi hàm cần đo và đặt mốc kết thúc thời gian đo

Sau khi hàm thực thi xong, nhóm lại gọi time.time() để lấy thời gian hiện tại và lưu giá trị này vào một biến khác, thường gọi là end_time.

Bước 3: Tính thời gian thực thi:

```

execution_time = end_time - start_time
# In ra kết quả
print(f"Execution time: {execution_time} seconds")
```

Hình 2-27 Tính toán thời gian thực thi

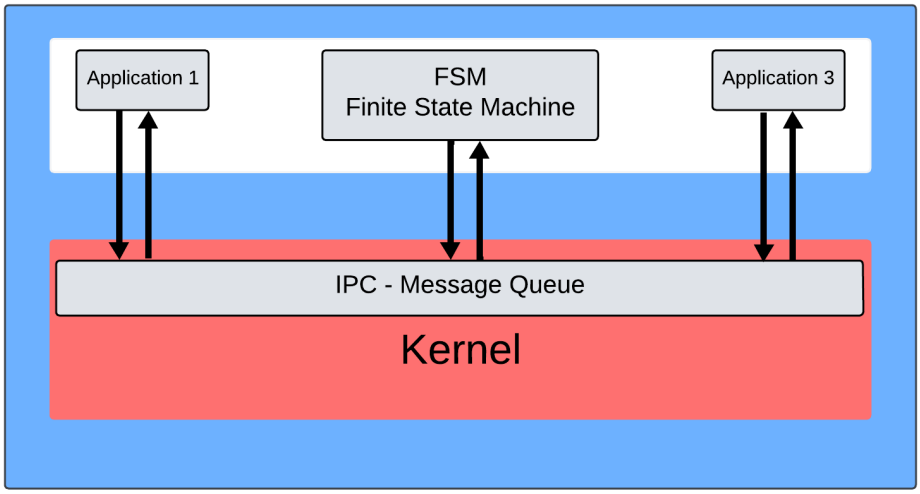
Nhóm tính toán thời gian đã trôi qua bằng cách lấy end_time trừ đi start_time. Kết quả là khoảng thời gian thực thi của hàm và được tính bằng giây.

Hạn chế khi sử dụng hàm time.time:

+ Độ chính xác hạn chế: time.time() có độ chính xác chỉ đến đơn vị giây và phần lẻ của giây, có thể không đủ chính xác cho các tác vụ cần đo lường với độ chính xác cao hơn.
+ Bị ảnh hưởng bởi điều chỉnh hệ thống: Thời gian được lấy từ hệ thống có thể bị ảnh hưởng bởi các điều chỉnh thời gian hệ thống như đồng bộ thời gian mạng.

2.9 FRAMEWORK QUẢN LÝ HỆ THỐNG NHÚNG

2.9.1 Lý thuyết về framework



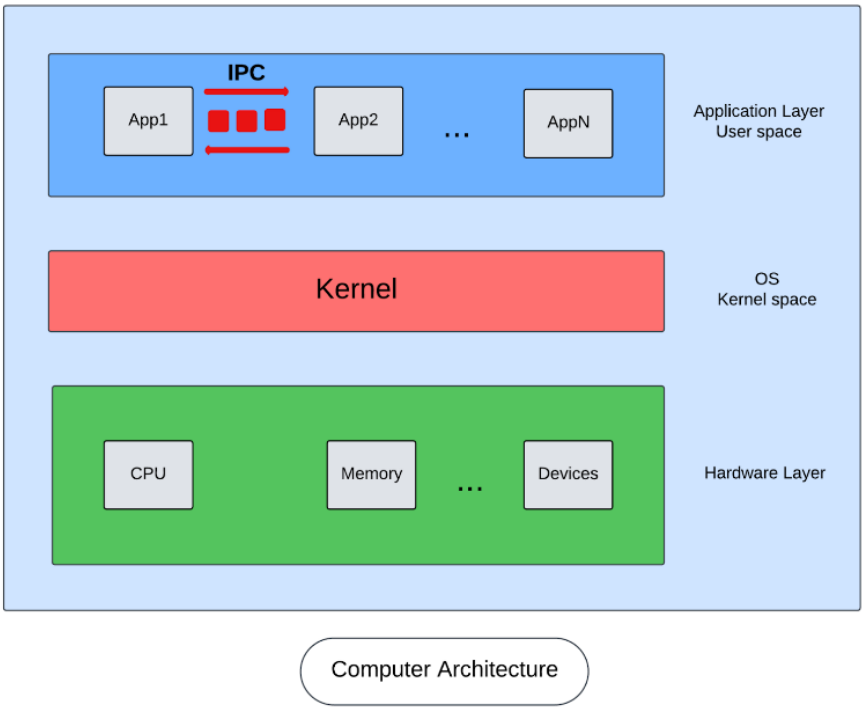
Hình 2-28 Mô hình FrameWork

Khi xây dựng lớp ứng dụng trong một hệ thống nhúng thì vấn đề quản lý các tiến trình để chúng hoạt động chính xác, ổn định và độ trễ thấp luôn được ưu tiên vì thế nhóm thực hiện đề tài đề xuất một kiến trúc quản lý hệ thống gồm 3 phần:

+ Phần ứng dụng: cung cấp giao diện tương tác với người dùng.
+ Phần FSM: chịu trách nhiệm quản lý hệ thống và điều khiển thay đổi trạng thái.
+ Phần IPC: là cầu nối giữa phần ứng dụng và phần FSM, chịu trách nhiệm phân luồng dữ liệu để truyền đạt thông tin.

2.9.2 Sơ lược về IPC

IPC (Inter Process Communication) là một cơ chế sử dụng hai hoặc nhiều tiến trình chạy trên cùng một máy để trao đổi dữ liệu với nhau.



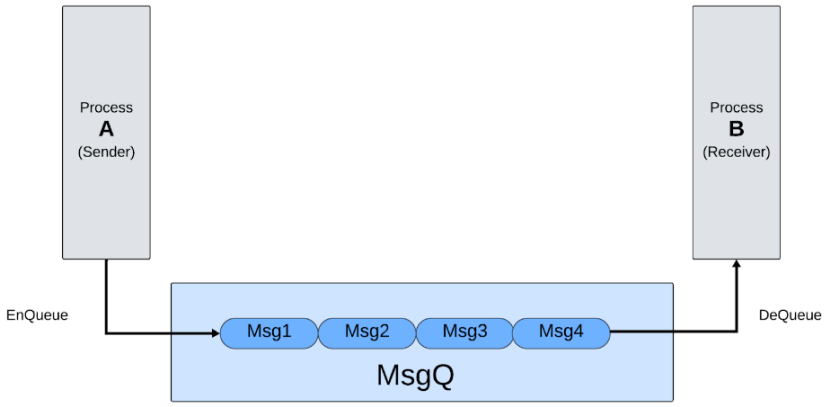
Hình 2-29 Kiến trúc máy tính

Kiến trúc máy tính thường sẽ có 3 lớp cơ bản:

+ Lớp Application: bất kì ứng dụng nào được thực thi trên máy tính hoặc bất kì phần mềm nào được cài được điều sẽ được chạy ở lớp này.
+ Lớp Operating system hoặc Kernel space: đây là nơi hệ điều hành hoạt động.
+ Lớp Hardware: bao gồm bất kì thành phần nào liên quan đến phần cứng như CPU, RAM và các thiết bị ngoại vi.

IPC sẽ giúp các ứng dụng trong lớp Application tương tác với nhau. IPC bao gồm 4 phương pháp: Unix Domain Sockets, Message Queues, Shared Memory và Signals. Trong đề tài này nhóm thực hiện đề tài quan tâm hơn về Message Queues.

* Lý thuyết của Message Queue



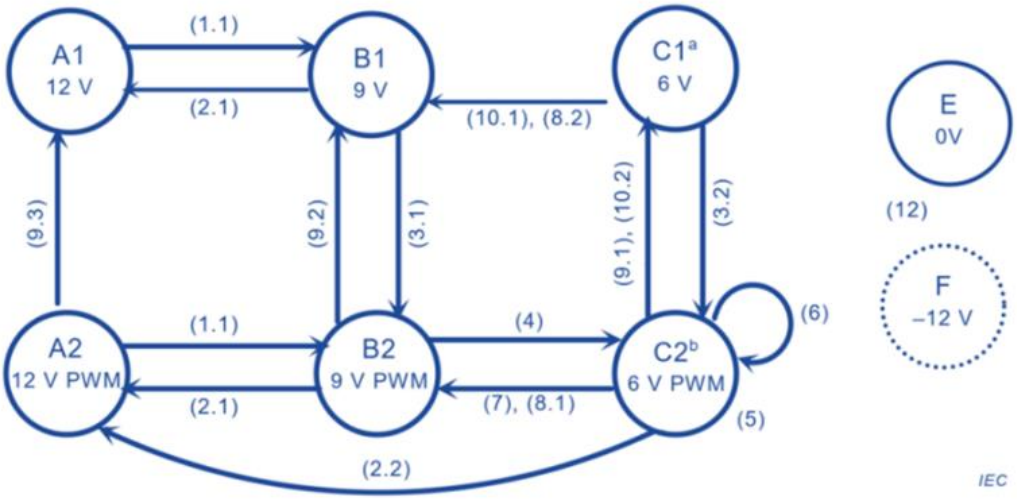
Hình 2-30 Mô hình hàng đợi tin nhắn

Những điểm khái quát về hàng đợi tin nhắn (Message Queue):

+ Hàng đợi tin nhắn là một danh sách liên kết các tin nhắn được lưu trữ trong kernel và được xác định bởi ID của hàng đợi tin nhắn đó. Một Hàng đợi tin nhắn được định nghĩa bằng ID và không được trùng ID với Hàng đợi tin nhắn khác cùng thời điểm.
+ Tin nhắn mới được thêm vào cuối hàng đợi và mọi tin nhắn đều có trường là số nguyên dương và độ dài không âm.
+ Tiến trình tạo một Hàng đợi tin nhắn hoặc sử dụng một Hàng đợi tin nhắn đã được tạo bởi một tiến trình khác.
+ Message queue được quản lý bởi Kernel/OS.
+ Tất cả các tiến trình có thể trao đổi thông tin thông qua quyền truy cập của hệ thống.
+ Chúng ta không cần phải lấy dữ liệu theo FIFO, thay vào đó chúng ta có thể lấy tin nhắn dựa vào trường loại của chúng [23].

2.9.3 Sơ lược về FSM

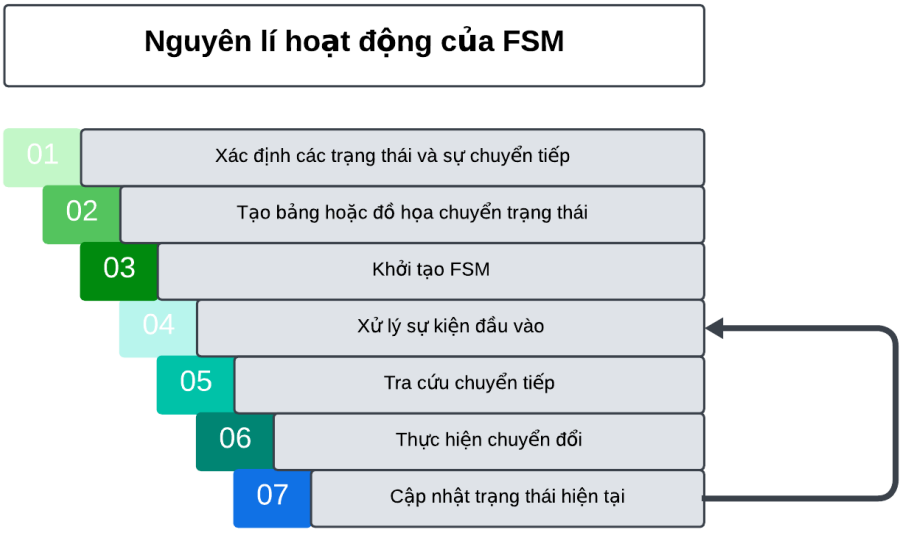
FSM (Finite State Machine) - máy trạng thái hữu hạn là một mô hình học biểu diễn trạng thái của hệ thống, trong đó số trạng thái là hữu hạn. Từ mỗi trạng thái, máy có thể chuyển đổi qua một trạng thái cố định khác dựa vào các sự kiện hoặc điều kiện. Các hệ thống có hành vi tuần tự và rời rạc có thể được thiết kế, phân tích và triển khai bằng cách sử dụng mô hình này.



Hình 2-31 Mô hình trạng thái của trạm sạc xe điện

Hình 2-28 là các trạng thái của trạm sạc xe điện với A1, A2, B1, B2,... là các trạng thái và 1.1, 2.1, 3.1, …. là các điều kiện (IEC-61851-1) [24] gây ra việc chuyển đổi trạng thái. Khi không sử dụng FSM thì sẽ gây ra các vấn đề sau:

+ Code dài, khó mở rộng, dễ xảy ra lỗi, độ trễ cao.
+ Khó bảo trì và cập nhật.
+ Khó quản lý để đảm bảo luồng dữ liệu chạy đúng.



Hình 2-32 Nguyên lý hoạt động của FSM

Nhờ cơ chế này mà FSM sẽ giúp chúng ta mô tả các trạng thái, sự kiện và quá trình chuyển đổi giữa các trạng thái từ đó hệ thống sẽ hoạt động hiệu quả hơn và có thể tái sử dụng trong nhiều ứng dụng khác nhau [25].

2.10 JETSON NANO

2.10.1 Giới thiệu về Jetson Nano

NVIDIA Jetson Nano Developer Kit mang đến hiệu suất tính toán để chạy các công việc liên quan đến AI với kích thước, năng lượng và chi phí tốt. Giúp các nhà phát triển có thể chạy các khung và mô hình AI cho các ứng dụng như phân loại hình ảnh, phát hiện đối tượng, phân đoạn và xử lý giọng nói. Bộ công cụ phát triển này có thể được cấp nguồn bằng micro-USB và đi kèm với nhiều I/O, từ GPIO đến CSI, giúp các nhà phát triển dễ dàng kết nối nhiều cảm biến mới để kích hoạt các ứng dụng AI đa dạng. Jetson Nano cực kỳ tiết kiệm năng lượng, chỉ tiêu thụ 5W.

Hình 2-33 Board Jetson Nano

Jetson Nano cũng được hỗ trợ bởi NVIDIA JetPack, bao gồm gói hỗ trợ bảng mạch (BSP), hệ điều hành Linux, các thư viện phần mềm NVIDIA CUDA, cuDNN và TensorRT cho học sâu, thị giác máy tính, tính toán GPU, xử lý đa phương tiện và nhiều hơn nữa. Phần mềm này có sẵn thông qua hình ảnh thẻ SD dễ dàng để flash, giúp khởi động nhanh chóng và dễ dàng. Cùng một SDK JetPack được sử dụng trên toàn bộ dòng sản phẩm NVIDIA Jetson và hoàn toàn tương thích với nền tảng AI hàng đầu thế giới của NVIDIA cho việc đào tạo và triển khai phần mềm AI. Bộ phần mềm đã được chứng minh này giảm thiểu sự phức tạp và nỗ lực tổng thể cho các nhà phát triển.

2.10.2 So sánh Jetson Nano và Raspberry Pi

Jetson Nano: Phù hợp cho các dự án AI, học máy, và các ứng dụng yêu cầu xử lý hình ảnh cao cấp. Ưu điểm chính là khả năng xử lý mạnh mẽ nhờ GPU CUDA của NVIDIA [26].

Raspberry Pi: Lý tưởng cho các dự án đa dạng, từ học tập, giáo dục đến các ứng dụng IoT, nhúng. Có cộng đồng lớn và giá cả hợp lý, phù hợp cho người mới bắt đầu.

Dưới đây là bảng so sánh một số tiêu chí giữa Jetson Nano và Raspberry Pi [27].

Bảng 2-2 So sánh Jetson Nano Developer Kit A02 với Raspberry Pi 4 Model B

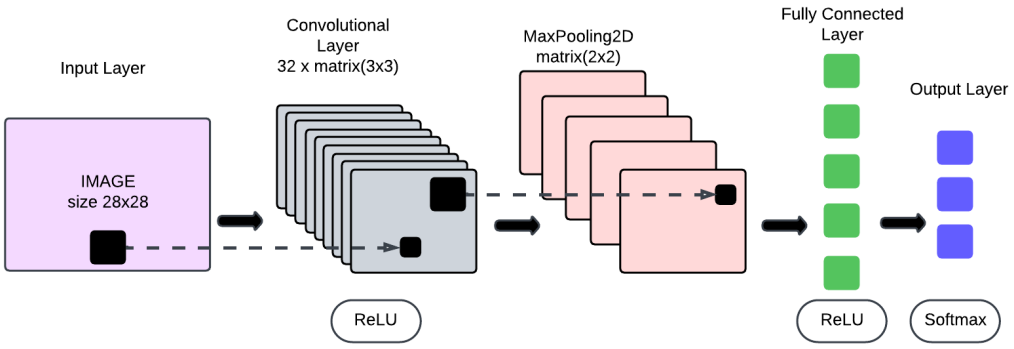
Tiêu chí	Jetson Nano Developer Kit A02	Raspberry Pi 4 Model B (4GB RAM)
GPU	NVIDIA Maxwell với 128 lõi CUDA	VideoCore VI
CPU	ARM Cortex-A57 4 nhân	ARM Cortex-A72 4 nhân
RAM	4GB LPDDR4	4GB LPDDR4
Ứng dụng AI và máy học	Tối ưu cho AI, hỗ trợ CUDA, cuDNN, TensorRT	Có thể chạy AI nhưng không tối ưu bằng Jetson Nano
Hệ sinh thái và cộng đồng	Công cụ phát triển mạnh mẽ từ NVIDIA, cộng đồng đang phát triển	Cộng đồng lớn, tài liệu phong phú, hỗ trợ tốt từ cộng đồng
Khả năng kết nối	4x USB 3.0, GPIO, MIPI-CSI, Ethernet Gigabit	2x USB 2.0, 2x USB 3.0, GPIO, 2x HDMI, cổng camera CSI và DSI, Ethernet Gigabit, Wi-Fi

Vì những ưu điểm và thông số kỹ thuật trên nên nhóm thực hiện đề tài đã chọn Jetson Nano Developer Kit A02 cho nghiên cứu lần này.

CHƯƠNG 3
XÂY DỰNG MÔ HÌNH VÀ THIẾT KẾ HỆ THỐNG

3.1 MÔ HÌNH CNN

Trong phần này, nhóm thực hiện đề tài sẽ trình bày kiến trúc mô hình học sâu phổ biến và mạnh mẽ trong lĩnh vực xử lý hình ảnh - mạng nơ-ron tích chập (Convolutional Neural Network - CNN). CNN đã trở thành trụ cột trong nhiều ứng dụng liên quan đến thị giác máy tính như nhận dạng hình ảnh, phát hiện đối tượng, và phân loại hình ảnh. Để minh họa cho cấu trúc và cách hoạt động của mô hình CNN, hãy cùng xem xét hình ảnh dưới đây.



Hình 3-1 Mô hình CNN

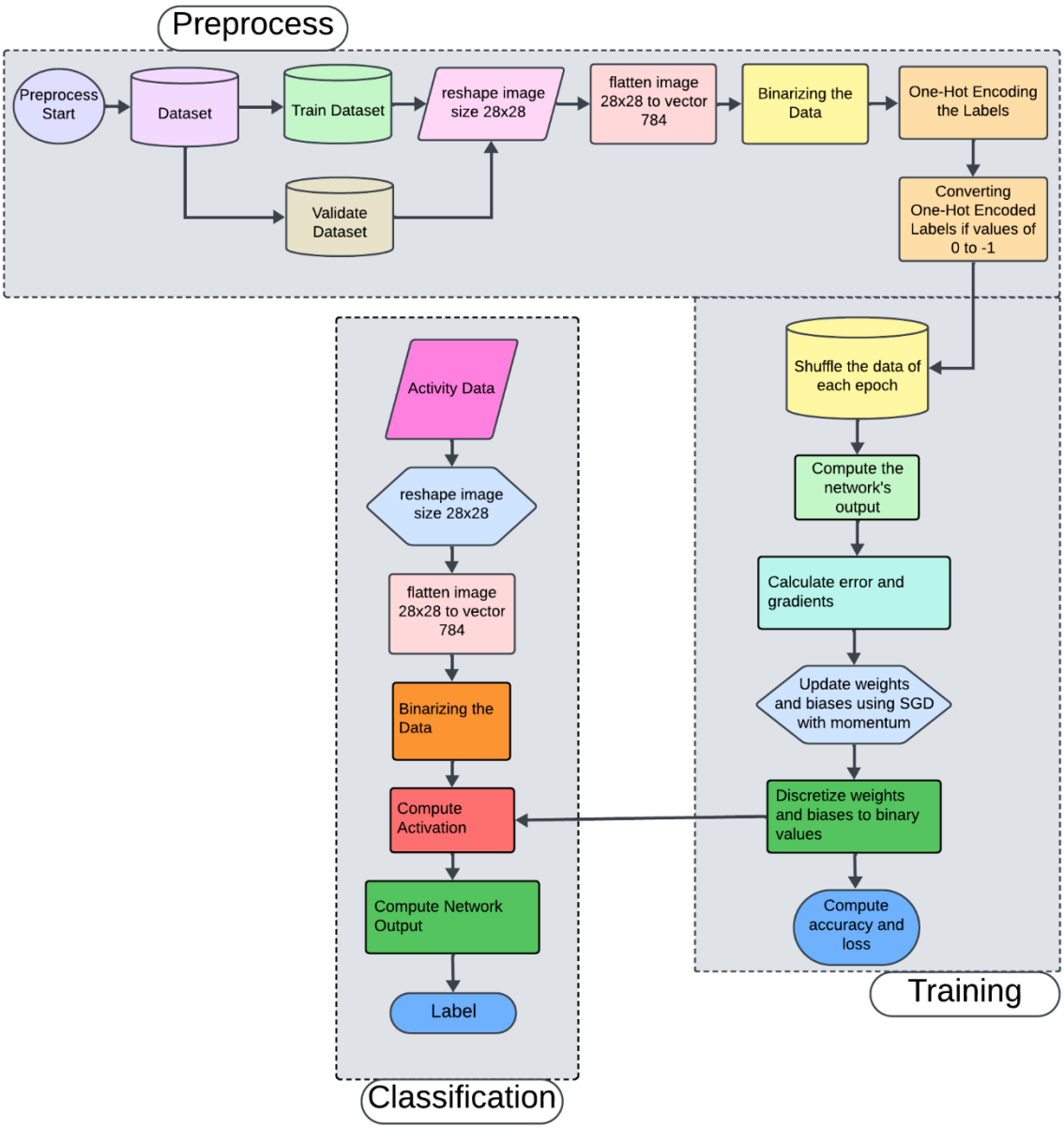
Mô hình Convolutional Neural Network (CNN) này bao gồm các lớp chính: lớp Convolutional (Conv2D) với 32 filters kích thước 3x3 và hàm kích hoạt ReLU, lớp Pooling (MaxPooling2D) với kích thước 2x2, sau đó dữ liệu sẽ được làm phẳng hóa (Flatten) để chuyển đổi dữ liệu từ ma trận nhiều chiều sang một vector 1 chiều, lớp Fully Connected với 512 đơn vị và hàm kích hoạt ReLU, và lớp Đầu ra (Output Layer) với 10 đơn vị và hàm kích hoạt softmax. Lớp Convolutional giúp phát hiện các đặc trưng không gian từ ảnh đầu vào, trong khi lớp MaxPooling giảm kích thước không gian của đặc trưng, giúp mô hình mạnh mẽ hơn với các biến đổi của ảnh. Lớp Fully Connected tạo ra các kết nối giữa các nơ-ron, giúp mô hình học được các mẫu phức tạp hơn từ các đặc trưng đã phát

hiện cùng với hàm softmax giúp chuyển đổi đầu ra thành xác suất dự đoán cho từng lớp, thuận tiện cho bài toán phân loại.

3.2 KHẢO SÁT MÔ HÌNH BNN

3.2.1 Sơ đồ khối

Tiếp theo, nhóm thực hiện đề tài sẽ trình bày mô hình mạng nơ-ron nhị phân (Binary Neural Network - BNN). BNN là một biến thể của mạng nơ-ron truyền thống, trong đó trọng số và kích hoạt của các nơ-ron được biểu diễn dưới dạng nhị phân, giúp giảm thiểu đáng kể yêu cầu về tính toán và bộ nhớ. Điều này làm cho BNN trở thành lựa chọn lý tưởng cho các ứng dụng trên thiết bị di động và hệ thống nhúng. Hãy cùng xem xét sơ đồ khối mô hình BNN dưới đây để hiểu rõ hơn về cấu trúc, cơ chế hoạt động của nó và cách triển khai của nó.



Hình 3-2 Sơ đồ khối mô hình BNN

** Khối tiền xử lý:*

Quá trình preprocess dữ liệu trong bài toán này nhằm chuẩn bị dữ liệu cho việc huấn luyện mạng nơ-ron nhị phân (Binary Neural Network - BNN). Đầu tiên, bộ dữ liệu chứa các hình ảnh được đưa vào. Các hình ảnh này có kích thước 28x28 pixel và được chuyển thành các vector một chiều với 784 phần tử để dễ dàng xử lý trong mạng nơ-ron.

Tại khối Binarizing the Data, các giá trị pixel mức xám được chuyển đổi thành giá trị nhị phân (0 hoặc 1) thông qua việc sử dụng ngưỡng. Theo thang

mức xám thì giá trị 127 là giá trị trung bình nên nhóm thực hiện đề tài chọn 127 làm ngưỡng. Mỗi pixel trong hình ảnh được đặt thành 0 nếu giá trị pixel là nhỏ hơn hoặc bằng ngưỡng và là 1 nếu lớn hơn ngưỡng. Quá trình này giúp giảm độ phức tạp của dữ liệu, làm giảm nhiễu và tăng cường độ tương phản, làm cho dữ liệu phù hợp với mạng nơ-ron nhị phân.

Khối One-Hot Encoding the labels, thực hiện biến đổi các nhãn thành các vectơ nhị phân, trong đó chỉ có một phần tử là 1 (được gọi là "hot") và các phần tử còn lại là 0. Ví dụ, nhãn 3 sẽ được biểu diễn thành vectơ [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]. Quá trình này là cần thiết để đưa dữ liệu nhãn vào mạng nơ-ron và phản hồi đúng kết quả dự đoán. Bằng cách này, mô hình có thể học và dự đoán các nhãn một cách chính xác. Điều quan trọng là quá trình "one-hot encoding" đảm bảo rằng không có mối quan hệ số học (thứ tự) nào giữa các nhãn, giúp mô hình học mối quan hệ giữa chúng một cách chính xác.

Tiếp theo, nhãn của dữ liệu cũng được chuyển đổi thành dạng nhị phân với các nhãn -1 cho giá trị không và 1 cho giá trị tương ứng. Khi sử dụng -1 và 1, phép toán nhân sẽ trở nên đơn giản hơn vì nó tương đương với việc sử dụng phép toán logic AND với các giá trị nhị phân. Điều này có thể giúp giảm độ phức tạp của các phép toán và làm tăng hiệu suất tính toán, giảm thiểu độ chệch (bias).

** Khối huấn luyện:*

Quá trình huấn luyện mạng nơ-ron nhị phân (Binary Neural Network - BNN) bao gồm nhiều bước nhằm tối ưu hóa các tham số của mạng để đạt được độ chính xác cao nhất. Dữ liệu huấn luyện phải được xáo trộn để đảm bảo rằng mô hình không học được các mẫu cụ thể và phát triển mối quan hệ tổng quát giữa các đặc trưng và nhãn. Việc này giúp tránh overfitting bằng cách tạo ra tính ngẫu nhiên và giảm sự phụ thuộc vào thứ tự dữ liệu, giúp mạng nơ-ron học được các biểu diễn tốt nhất cho dữ liệu và cải thiện hiệu suất của mô hình. Sau đó chia thành các lô (batch) nhỏ để áp dụng thuật toán Gradient Descent. Trong mỗi lô, đầu tiên là tính toán giá trị kích hoạt của các nút trong lớp ẩn và lớp đầu ra bằng cách nhân ma trận đầu vào với trọng số và thêm vào độ lệch (bias). Sau đó, sai số giữa giá trị dự đoán và giá trị thực tế được tính toán và sử dụng để điều chỉnh trọng số

theo hướng giảm thiểu sai số này. Các cập nhật trọng số được thực hiện với một hệ số Momentum để giảm thiểu dao động và tăng tốc độ hội tụ. Cuối cùng, các trọng số và độ lệch được lượng tử hóa về nhị phân để phù hợp với kiến trúc của BNN. Quá trình này giúp mô hình học từ dữ liệu, điều chỉnh các tham số để cải thiện độ chính xác, và đảm bảo rằng mô hình vẫn giữ được tính đơn giản và hiệu quả của mạng nhị phân.

** Khối xử lý phân loại:*

Quá trình kiểm thử mạng nơ-ron nhị phân (Binary Neural Network - BNN) nhằm đánh giá hiệu suất của mô hình trên dữ liệu chưa từng thấy để đảm bảo tính tổng quát hóa và khả năng phân loại chính xác. Đầu tiên, mô hình đã huấn luyện được tải lên với các trọng số và độ lệch tốt nhất. Sau đó, mỗi hình ảnh trong bộ dữ liệu kiểm thử được xử lý qua các bước: thay đổi kích thước thành 28x28 pixel và lượng tử hóa các giá trị pixel thành dạng nhị phân. Tiếp theo, hình ảnh đã xử lý được đưa vào mô hình để dự đoán nhãn. Nhãn dự đoán này được so sánh với nhãn thực tế để tính toán độ chính xác của mô hình. Việc này giúp xác định mức độ chính xác của mô hình trên dữ liệu mới, đảm bảo rằng mô hình không chỉ hoạt động tốt trên dữ liệu huấn luyện mà còn có khả năng tổng quát hóa tốt trên các dữ liệu khác. Điều này rất quan trọng trong việc đánh giá hiệu quả và độ tin cậy của mô hình trước khi triển khai vào thực tế.

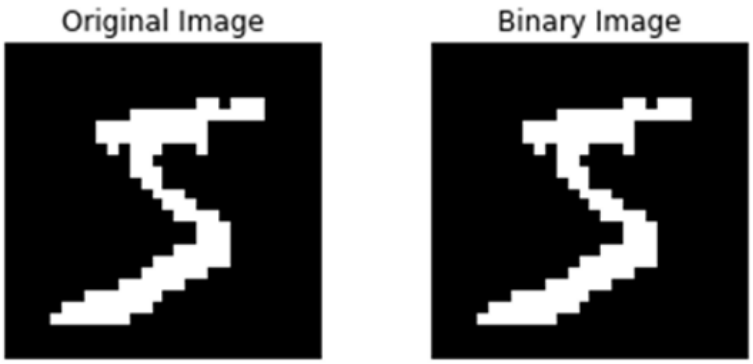
3.2.2 Mã giả

** Tiền xử lý:*

Input: train_dataset
Output: x_train, y_train, x_test, y_test
Function Preprocess(train_dataset): x_train = np.reshape(x_train, (60000, 784))) x_test = np.reshape(x_test, (10000, 784)) x_train[x_train <= 127] = 0 x_train[x_train > 127] = 1 x_test[x_test <= 127] = 0 x_test[x_test > 127] = 1 y_train = np.matrix(np.eye(10)[y_train]) y_test = np.matrix(np.eye(10)[y_test]) y_train[y_train == 0] = -1 y_test[y_test == 0] = -1

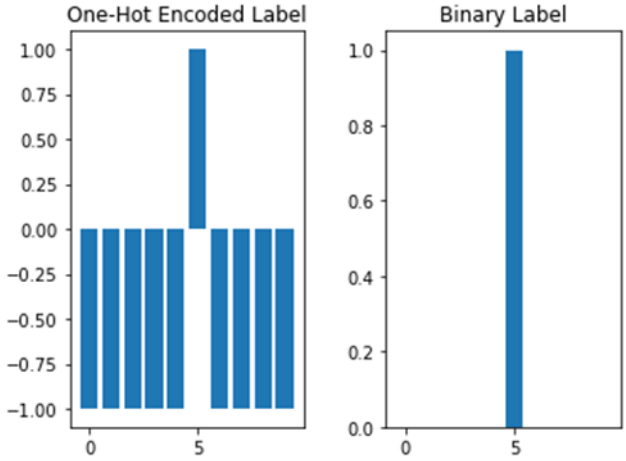
Hình thực hiện các bước tiền xử lý dữ liệu cho bộ dữ liệu MNIST trước khi huấn luyện mạng BNN. Đầu tiên, nó nhận bộ dữ liệu và chia thành dữ liệu huấn luyện và dữ liệu kiểm tra.

Sau đó, ảnh được chuyển đổi thành dạng phù hợp để phù hợp với mạng nơ-ron, và dữ liệu được chuyển đổi thành dạng nhị phân để đơn giản hóa quá trình huấn luyện.



Hình 3-3 Hiệu chỉnh kích thước ảnh

Tiếp theo, nhãn được mã hóa dưới dạng one-hot encoding và chuyển đổi thành dạng nhị phân (-1/1). Kết quả là một tập dữ liệu sẵn sàng cho việc huấn luyện mạng nơ-ron nhị phân để nhận dạng chữ số viết tay.



Hình 3-4 Mã hóa và nhị phân hóa nhãn

** Huấn luyện:*

Input: x_train, y_train
Output: Wh, bh, Wo, bo
Function Train(train_dataset): random.shuffle(SampleIdx) for i in range(0, number_sample - size_Batch, size_Batch): sample_Batch = SampleIdx[i:i + size_Batch] x = matrix(x_train[sample_Batch, :]) y = matrix(y_train[sample_Batch, :]) a = sign(np.dot(x, Wh.T) + bh) o = sign(np.dot(a, Wo.T) + bo) Cost[IdxCost] = mean(mean(power((y - o), 2), axis=1)) IdxCost += 1 do = (y - o) dWo = matrix(np.dot(do.T, a) / size_Batch) dbo = mean(do, 0) WoUpdate = LearningRate * dWo + momentum * del_Wo

```
    boUpdate = LearningRate * dbo + momentum * del_bo
    del_Wo = Wo_update
    del_bo = bo_update
    dh = np.dot(do, Wo)
    dWh = np.dot(dh.T, x) / size_Batch
    dbh = mean(dh, 0)

    WhUpdate = LearningRate * dWh + momentum * del_Wh
    bhUpdate = LearningRate * dbh + momentum * del_bh
    del_Wh = Wh_update
    del_bh = bh_update
    Wo = Wo + Wo_update
    bo = bo + bo_update
    Wh = Wh + Wh_update
    bh = bh +  bh_update
    Wo = sign(Wo)
    bo = sign(bo)
    Wh = sign(Wh)
    bh = sign(bh)
MSE[ep] = mean(Cost)
IdxCost = 0

feed_forward = resources.FeedForward(x_test, Wh, bh, Wo, bo)
BiOutN = feed_forward.OutN
acc_test = resources.AccTest(BiOutN, y_test)
BiAccuracy = acc_test.accuracy
BiAcc[ep] = BiAccuracy*100
```

Đoạn code này thực hiện huấn luyện một mạng BNN bằng phương pháp mini-batch gradient descent. Cụ thể, dữ liệu huấn luyện được xáo trộn và chia thành các mini-batch. Với mỗi mini-batch, đầu ra của mạng neural được tính dựa trên trọng số hiện tại và hàm kích hoạt. Sai số giữa đầu ra thực tế và dự đoán được sử dụng để tính gradient của hàm mất mát theo các tham số mạng. Cập nhật trọng số của mạng theo hướng giảm gradient với một tốc độ học và một momentum. Các trọng số được giữ trong khoảng [-1, 1] bằng cách giữ dấu của chúng sau mỗi lần cập nhật. Độ lỗi trung bình (MSE) được tính sau mỗi epoch để đánh giá hiệu suất của mạng trên dữ liệu huấn luyện. Mạng sau mỗi epoch cũng được kiểm thử trên dữ liệu kiểm tra để đo lường độ chính xác nhị phân.

** Xử lý phân loại:*

Input: activity_datate
Output: predict_label
Function Test(activity_data, Wh, bh, wo, bo): image = activity_data.resize((28, 28)) image = image.convert('L') image = np.array(image) image = np.reshape(image, (1, 784)) image [image <= 127] = 0 image [image > 127] = 1 activity = np.sign(np.dot(image, Wh.T) + bh) output = np.sign(np.dot(a, Wo.T) + bo) predicted_label = np.argmax(output)

Trước khi đưa vào mạng BNN, hình ảnh được tiền xử lý bằng cách chuyển đổi về ảnh xám, thay đổi kích thước về 28x28 pixel và chuyển đổi sang dạng numpy array. Sau đó, các giá trị pixel được chuẩn hóa và nhị phân hóa để tạo ra đầu vào cho mạng BNN.

Quá trình dự đoán được thực hiện thông qua phép tính nhị phân, trong đó các trọng số và đầu vào đều được biểu diễn bằng các giá trị nhị phân (-1 hoặc +1). Các phép toán dot product (tích vô hướng) được thực hiện giữa đầu vào và các trọng số của lớp ẩn, sau đó kết quả được đưa qua hàm sign để biến đổi thành các giá trị nhị phân. Tương tự, các phép toán này được lặp lại cho lớp đầu ra để tạo ra dự đoán cuối cùng.

Kết quả dự đoán là một vectơ nhị phân, trong đó mỗi phần tử đại diện cho xác suất của một số từ 0 đến 9. Số được dự đoán là vị trí của phần tử có giá trị nhị phân lớn nhất trong vectơ này. Với thuật toán này, mạng BNN có khả năng dự đoán số chính xác từ hình ảnh đầu vào và làm việc hiệu quả trên dữ liệu kiểm tra từ tập dữ liệu MNIST.

** BitArray:*

Input: weight
Output: bit_weight
Function BitArray(weight): binary_ weight = np.where(weight > 0, 1, 0) flattened_ binary_ weight = binary_ weight.flatten() bit_weight = bitarray(flattened_ binary_ weight.tolist()) return bit_weight

Khi áp dụng thuật toán BitArray vào BNN. Sau khi huấn luyện kết thúc, các trọng số tốt nhất mà mô hình đạt được sẽ đưa vào hàm BitArray. Tại đây, trọng số được nhị phân hóa, làm phẳng và cuối cùng là chuyển đổi về bitArray. Kết quả là ta sẽ được một mảng bit nhị phân để lưu trữ.

** Phân cụm với K-Means:*

Input: dataset, cluster_num
Output: preprocessed_dataset

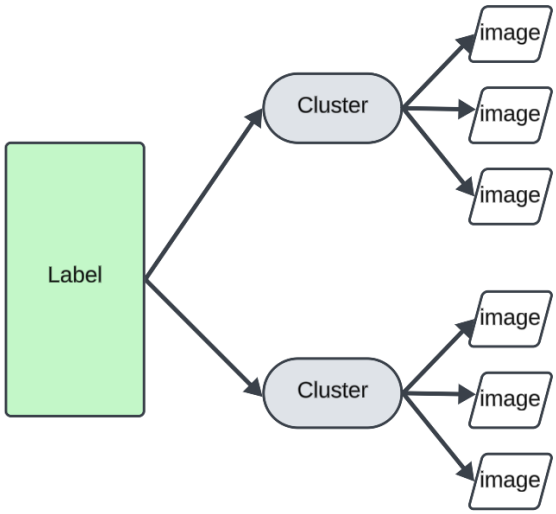
Function Cluster(dataset, cluster_num):
for label, images in dataset.items():
if len(images) >= cluster_num:
features = np.array(get_feature(images))
model = KMeans(cluster_num, n_init=10)
model.fit(features)
preds = model.predict(features)
for i, img in enumerate(images):
cluster = preds[i]
preprocessed_dataset[label][cluster].append(img)
else:
single_cluster = 0
for img in images:
preprocessed_dataset[label][single_cluster].append(img)
return preprocessed_dataset

Nhóm vừa trình bày mã giả cho thuật toán là phân cụm. Phương pháp này giúp tổ chức dữ liệu thành các nhóm có tính chất tương tự dựa trên các đặc trưng của hình ảnh.

Thuật toán bắt đầu bằng việc duyệt qua từng nhãn trong bộ dữ liệu hình ảnh. Sau đó, nó kiểm tra số lượng hình ảnh trong từng nhãn và quyết định liệu có đủ để thực hiện phân cụm hay không. Nếu có, nó trích xuất các đặc trưng từ hình ảnh và sử dụng thuật toán K-Means để phân cụm chúng thành số lượng cụm được chỉ định. Cuối cùng, mỗi hình ảnh được gán vào một cụm tương ứng.

Việc này giúp tạo ra một tập dữ liệu như hình 3-5 với các cụm hình ảnh có tính chất tương tự, giúp cho việc phân tích và xử lý dữ liệu trở nên dễ dàng hơn. Cụ thể, trong ứng dụng thực tế, việc này có thể được sử dụng để tự động phân

loại hình ảnh, nhận dạng đối tượng hoặc thậm chí là tạo ra các tập dữ liệu được tổ chức sẵn cho việc huấn luyện mô hình học máy.

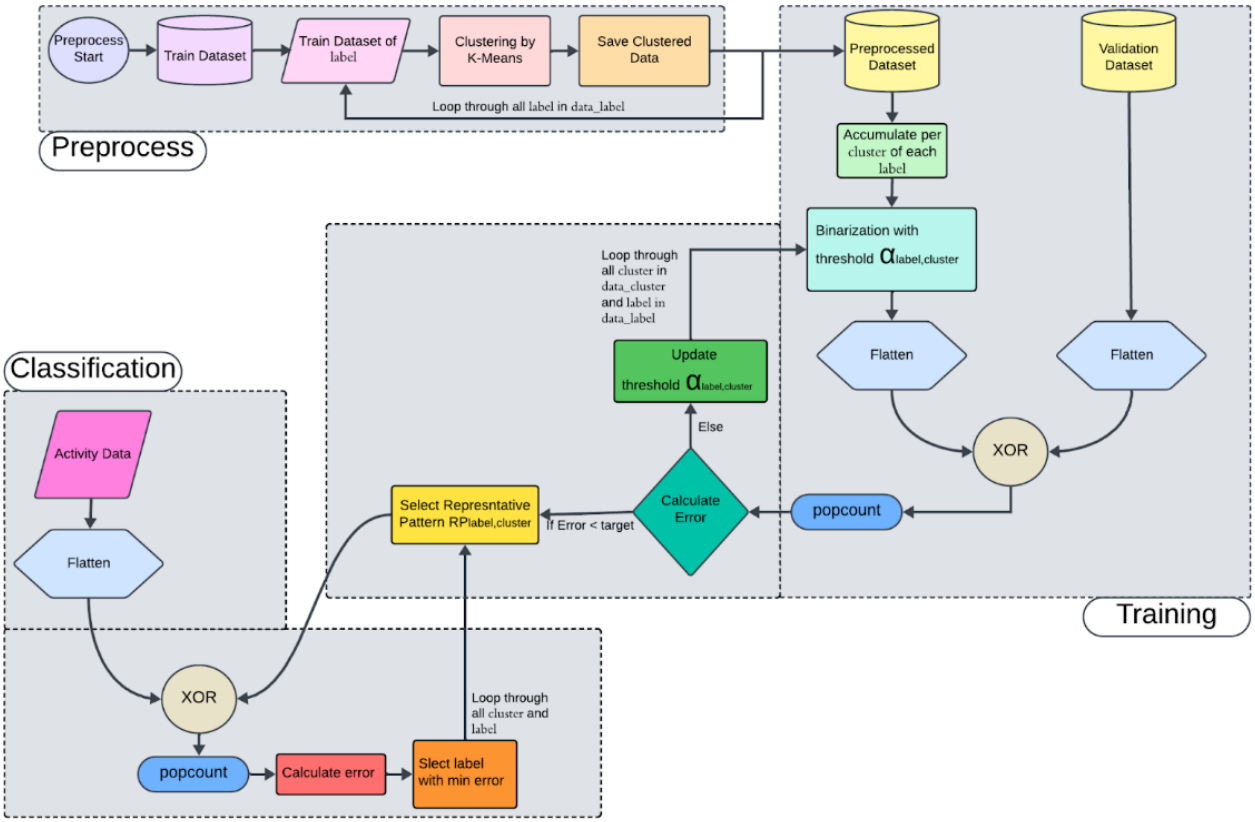


Hình 3-5 Mô hình phân cụm với K-Means

3.3 KHẢO SÁT MÔ HÌNH LSVM KẾT HỢP VỚI K-MEANS

3.3.1 Sơ đồ khối

Cuối cùng, nhóm thực hiện đề tài sẽ trình bày mô hình Linear Support Vector Machine (LSVM). LSVM là một phương pháp phân loại tuyến tính được sử dụng rộng rãi trong nhiều lĩnh vực, từ nhận dạng chữ viết tay đến phân loại văn bản. Mô hình này hoạt động bằng cách tìm ra một siêu phẳng tối ưu để phân chia dữ liệu thành các lớp khác nhau. Để giúp bạn hình dung rõ hơn về cấu trúc và nguyên lý hoạt động của LSVM, hãy cùng xem xét sơ đồ khối dưới đây.



Hình 3-6 Sơ đồ khối mô hình LSVM

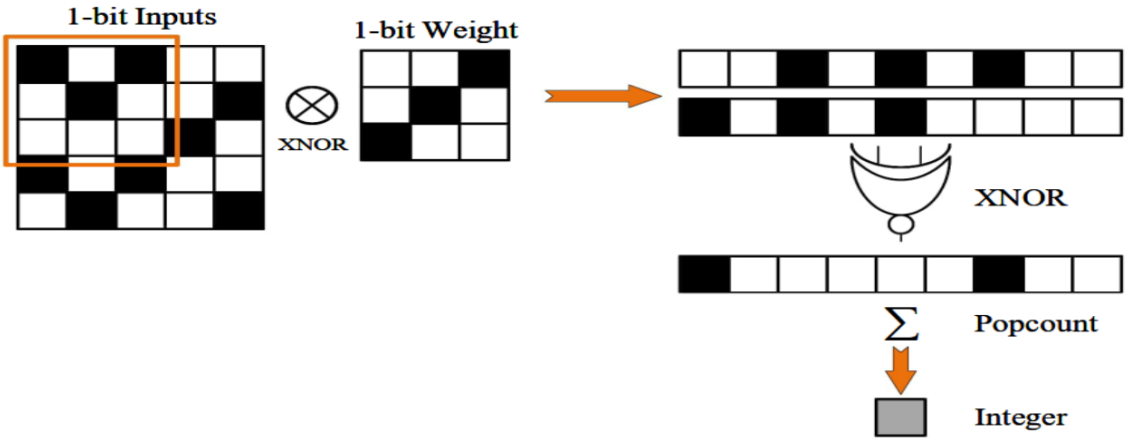
** Tiền xử lý:*

Quá trình tiền xử lý dữ liệu rất quan trọng trong việc tạo ra các siêu phẳng hiệu quả. Nhóm thực hiện đề tài đã tổng hợp và nhị phân hóa tập dữ liệu huấn luyện để tạo ra các siêu phẳng. Việc đơn giản tích lũy và nhị phân hóa dữ liệu có thể giảm kích thước dữ liệu trong khi vẫn giữ nguyên các đặc điểm tổng quát của dữ liệu. Tuy nhiên, hình ảnh sẽ mất đi các đặc điểm chi tiết của dữ liệu. Nhóm thực hiện đề tài nhận thấy rằng các đặc điểm chi tiết này có ảnh hưởng đáng kể đến việc phân loại. Nhóm thực hiện đề tài cũng quan sát thấy rằng các hình ảnh trong cùng một nhãn của tập dữ liệu MNIST có những đặc điểm hơi khác nhau. Ví dụ, một số chữ viết tay là chữ nghiêng, trong khi một số khác là chữ đậm. Để giải quyết vấn đề này, nhóm thực hiện đề tài đã thực hiện phân cụm dữ liệu huấn luyện của cùng một nhãn để làm cho các siêu phẳng của nhóm thực hiện đề tài đại diện cho các đặc điểm chi tiết của dữ liệu.. Hàm Preprocess trả về dữ liệu đã phân cụm từ dữ liệu huấn luyện cho mỗi nhãn. Để nhóm dữ liệu từ cùng một

nhãn, đầu tiên nhóm thực hiện đề tài nhóm dữ liệu huấn luyện theo nhãn của chúng. Sau đó, nhóm thực hiện đề tài áp dụng phương pháp phân cụm K-Means cho dữ liệu đã được nhóm theo từng nhãn. Cuối cùng, dữ liệu thuộc cùng một cụm của cùng một nhãn được lưu trữ.

** Huấn luyện:*

Quá trình huấn luyện đề xuất cho việc tạo siêu phẳng bao gồm việc tạo mẫu đại diện cho mỗi cụm của từng nhãn từ dữ liệu đã tiền xử lý và tập dữ liệu xác thực. Nhóm thực hiện đề tài tích lũy dữ liệu tiền xử lý thành hình ảnh R và nhị phân hóa giá trị pixel theo ngưỡng α để giảm ảnh hưởng của các giá trị ngoại lệ, sau đó làm phẳng R thành mẫu đại diện RP. Tập dữ liệu xác thực của nhãn j được chia thành các cụm và áp dụng phép XOR giữa RP và td từ đó xác định lỗi giữa RP và td bằng cách đếm số lượng giá trị khác nhau trong phép XOR. Ngưỡng α được cập nhật hoặc quá trình huấn luyện kết thúc khi lỗi dưới mục tiêu. Các siêu phẳng này giúp mô hình phân loại chính xác hơn bằng cách nắm bắt các đặc điểm chi tiết của dữ liệu.



Hình 3-7 Quá trình XNOR popcount

** Xử lý phân loại:*

Trong quá trình phân loại, nhóm thực hiện đề tài sử dụng một thuật toán phân loại tốc độ cao dựa trên các hoạt động bitwise. 'Classify' là một hàm xác định nhãn của dữ liệu hoạt động đầu vào. Quá trình phân loại tương tự như việc tính toán lỗi trong thuật toán huấn luyện. Đầu tiên, nhóm thực hiện đề tài tính toán lỗi bằng cách áp dụng phép XOR và popcount giữa siêu phẳng đại diện RP

và dữ liệu đầu vào. Trên các bộ xử lý có nguồn lực thấp, các hoạt động bitwise thực hiện nhanh hơn so với phép nhân và cộng. Do đó, nhóm thực hiện đề tài có thể cải thiện tốc độ của thuật toán.

3.3.2 Mã giả

**Tiền xử lý:*

Input: train_dataset, cluster_num
Output: preprocessed_dataset
Function Preprocess(train_dataset,cluster_num): For label, images in train_dataset.items: features = array[get_feature each image] model = KMeans(cluster_num) preds = model.predict(features) For i, image in images: cluster = preds[i] preprocessed_dataset[label][cluster].append(image)

Tại tiền xử lý, nhóm thực hiện đề tài sẽ tiến hành trích xuất đặc trưng của từng ảnh bằng phương pháp tính cường độ sáng của ảnh sau đó dựa vào số cụm mong muốn, thuật toán K-Means sẽ phân loại ảnh thành các cụm khác với mỗi cụm sẽ mang một đặc trưng nhất định ví dụ như độ dày, độ nghiêng, độ cao, …. của ký tự. Cuối cùng thì nhóm những ảnh chung đặc trưng lại thành một cụm để huấn luyện.

** Huấn luyện:*

Input: validation_dataset, preprocessed_dataset
Output: representative_pattern
Function Train(validation_dataset,preprocessed_dataset,num_epochs): While(num_epochs): for label,cluster in preprocessed_dataset for image in preprocessed_dataset[label][cluster] count += 1

```
            img_sum += img
        img_average = img_sum / count
        alpha[label[cluster] = img_average
        R[label][cluster] = img_sum
        For pixel in R[label][cluster]
            if pixel > alpha[label[cluster]
                pixel = 1
            else
                pixel = 0
        RP[label][cluster] = flatten(R[label][cluster])
        For td in validation_dataset[label][cluster]
            td = flatten(td)
            error += popcount(RP[label][cluster] XOR td)
        update(alpha[label[cluster],error)
```

Tại huấn luyện mỗi epoch, mô hình sẽ duyệt qua từng cụm. Tại đây nhóm thực hiện đề tài cộng tất cả ảnh lại (cộng ma trận) tạo thành một ma trận mẫu R[label][cluster] sau đó tính ra ma trận trung bình để dùng nó làm ma trận ngưỡng alpha alpha[label[cluster] của cụm đó. Sau đó nhị phân hóa ma trận mẫu bằng cách xét xem phần tử trong ma trận mẫu lớn hơn phần tử ma trận ngưỡng tại vị trí tương ứng thì thay đổi giá trị bằng 1, ngược lại thì bằng 0. Cuối cùng thu được ma trận đại diện mới RP[label][cluster] và xor nó với từng ảnh trong cụm tương ứng của tập đánh giá, từ đó tính toán được lỗi và cập nhật lại ngưỡng.

** Xử lý phân loại:*

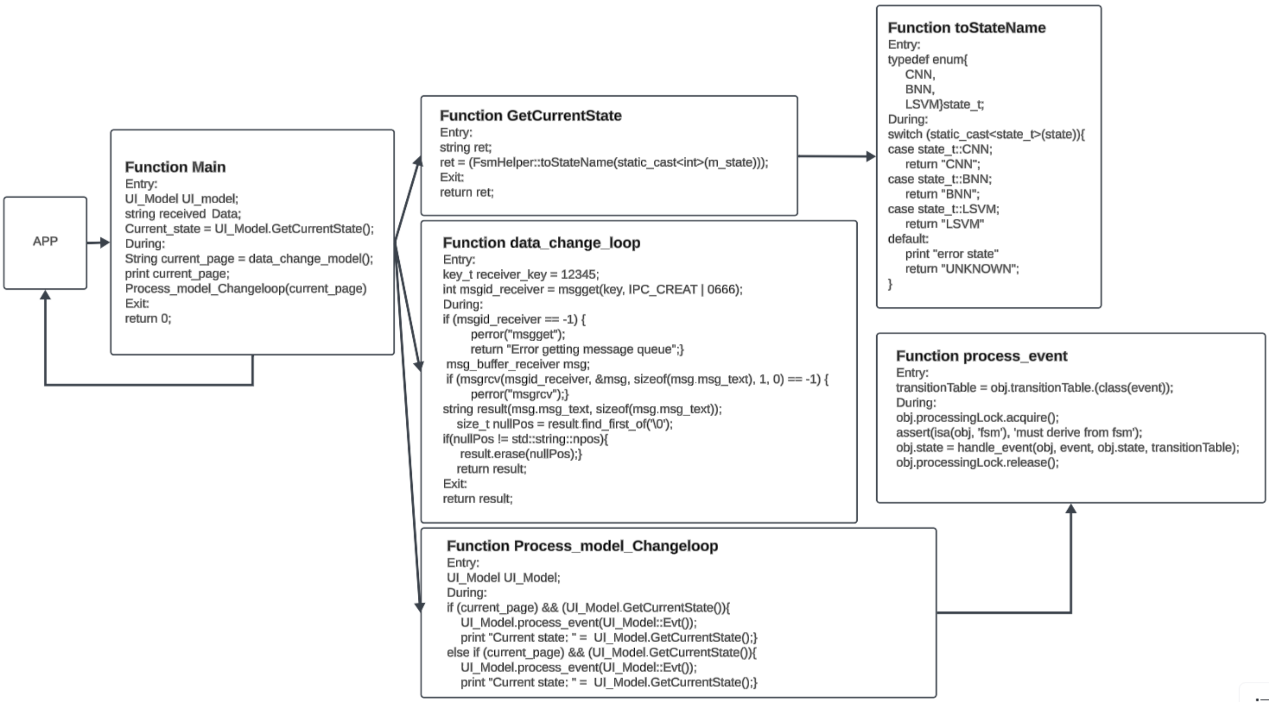
Input: activity_data
Output:determined_label
Function Classify(activity_data,representative_img,alpha): for label, cluster in representative_img: binary_activity_data = (activity_data > alpha[label][cluster, 1, 0) flatten_binary_activity_data = flatten(binary_activity_data) xor = popcount(flatten_binary_activity_data,representative_img[label][cluster] if error < min_error min_error = error determined_label = label

Tại xử lý phân loại, mô hình sẽ duyệt qua tất cả các cụm của từng nhãn, tại đây ảnh đầu vào sẽ được nhị phân hóa với ngưỡng alpha tương ứng sau đó sẽ thực hiện phép xor với ma trận đại diện của mỗi cụm, kết quả của cụm nào cho lỗi ít nhất thì kết quả dự đoán sẽ là nhãn của cụm đó [28].

3.4 THIẾT KẾ FRAMEWORK

3.4.1 FSM

Nhóm thực hiện đề tài sử dụng cách vẽ Stateflow giống trong MATLAB, để mô hình hóa hệ thống điều khiển phức tạp dựa trên trạng thái. Biểu đồ stateflow của FSM minh họa cách một hệ thống chuyển đổi giữa các trạng thái khác nhau dựa trên các điều kiện và sự kiện cụ thể. Dưới đây là biểu đồ stateflow của FSM, cho thấy cách hệ thống hoạt động thông qua các trạng thái và chuyển đổi tương ứng.



Hình 3-8 Biểu đồ stateflow của FSM

Hình 3-8 là cách vận hành của cơ chế FSM. FSM sẽ kiểm tra trạng thái hiện tại, sau đó nó đợi yêu cầu chuyển trạng thái từ APP thông qua message queue. Lúc này, hàm chuyển trạng thái và xử lý kiện khi chuyển trạng thái được gọi, nó sẽ kiểm tra xem trạng thái bắt đầu, hàm xử lý xử kiện và trạng thái kết thúc trong bảng transition_table ở hình 3-9 dưới đây.

```
using transition_table = table<
    //              Start                   Event                                   Target                          Action
    //---------+-------------------+-----------------------------------------+------------------------------------+-----------------
        row<         CNN,            Change_model_CNNtoBNN_Evt,                BNN,                            &m::BNN_Act
        row<         CNN,            Change_model_CNNtoLSVM_Evt,               LSVM,                           &m::LSVM_Act
        row<         BNN,            Change_model_BNNtoCNN_Evt,                CNN,                            &m::CNN_Act
        row<         BNN,            Change_model_BNNtoLSVM_Evt,               LSVM,                           &m::LSVM_Act
        row<         LSVM,           Change_model_LSVMtoCNN_Evt,               CNN,                            &m::CNN_Act
        row<         LSVM,           Change_model_LSVMtoBNN_Evt,               BNN,                            &m::BNN_Act
```

Hình 3-9 Khai báo trạng thái, sự kiện, hành động của FSM

Nhờ cách bố trí như này mà việc xác định trạng thái, xử lý sự kiện hoạt động đúng, hạn chế hệ thống bị treo. Bên cạnh đó, việc bảo trì cập nhật và tái sử dụng ở nhiều hệ thống khác nhau cũng khá hiệu quả.

3.5 PHƯƠNG PHÁP ĐÁNH GIÁ

Trong nghiên cứu này, để đánh giá hiệu suất của các mô hình, nhóm thực hiện đề tài sẽ sử dụng tập dữ liệu MNIST của Keras. Dữ liệu sẽ được phân chia thành tập huấn luyện và tập đánh giá. Riêng tập ảnh để phân loại thử nghiệm sẽ sử dụng một tập dữ liệu khác. Hiệu suất của các mô hình sẽ được đánh giá dựa trên các tiêu chí sau:

1. Độ chính xác (Accuracy):
    + Độ chính xác tổng thể: Tỷ lệ phần trăm các dự đoán đúng trên tổng số mẫu trong tập đánh giá.
    + Độ chính xác trên từng lớp: Đánh giá độ chính xác của mô hình trên từng lớp (chữ số) riêng lẻ để đảm bảo mô hình không chỉ tốt ở một số lớp nhất định.
2. Thời gian thực thi (Execution Time):
    + Thời gian huấn luyện: Thời gian cần thiết để huấn luyện mô hình trên tập dữ liệu huấn luyện.
    + Thời gian suy luận: Thời gian cần thiết để mô hình thực hiện dự đoán trên một mẫu hoặc một tập mẫu trong tập đánh giá.
    + Thời gian phản hồi: Đặc biệt quan trọng đối với các ứng dụng thời gian thực, thời gian cần để hệ thống đưa ra kết quả sau khi nhận được đầu vào.
3. Kích thước mô hình (Model Size):
    + Dung lượng lưu trữ: Kích thước của mô hình khi lưu trữ trên đĩa (thường được đo bằng MB hoặc GB).
    + Sử dụng bộ nhớ: Lượng bộ nhớ (RAM) cần thiết để tải và chạy mô hình.
4. Hiệu suất tính toán (Computational Efficiency):
    + Sử dụng memory: Đánh giá mức độ sử dụng memory trong quá trình huấn luyện và suy luận.

+ Tốc độ xử lý: Đánh giá tốc độ xử lý của mô hình khi chạy trên các phần cứng khác nhau, bao gồm cả máy tính cá nhân.

5. Tính khả thi và linh hoạt (Feasibility and Flexibility):
   + Khả năng triển khai: Đánh giá khả năng triển khai mô hình trên các nền tảng khác nhau như đám mây, thiết bị di động, và các hệ thống nhúng.
   + Dễ dàng tích hợp: Khả năng tích hợp mô hình vào các hệ thống hiện có hoặc các ứng dụng khác nhau.
6. Tính ổn định và độ tin cậy (Stability and Reliability):
   + Hiệu suất nhất quán: Đánh giá hiệu suất của mô hình trên các tập dữ liệu khác nhau để đảm bảo mô hình hoạt động ổn định và đáng tin cậy.
   + Khả năng xử lý ngoại lệ: Đánh giá khả năng của mô hình trong việc xử lý các trường hợp ngoại lệ hoặc các đầu vào không điển hình.
7. Khả năng mở rộng (Scalability):
   + Khả năng xử lý dữ liệu lớn: Đánh giá khả năng của mô hình khi làm việc với các tập dữ liệu lớn hơn.
   + Hiệu suất khi tăng số lượng mẫu: Khả năng duy trì hiệu suất khi số lượng mẫu trong tập dữ liệu tăng lên.

Thông qua các tiêu chí này, nhóm thực hiện đề tài có thể đánh giá toàn diện hiệu suất của các mô hình được triển khai, từ đó lựa chọn mô hình phù hợp nhất cho từng ứng dụng cụ thể.

CHƯƠNG 4

KẾT QUẢ

4.1 TIÊU CHÍ ĐÁNH GIÁ

Tiêu chí đánh giá của nhóm thực hiện đề tài sẽ dựa vào hiệu suất của các mô hình, về độ chính xác, thời gian thực thi và kích thước mô hình để từ đó làm nổi bật ứng dụng của các mô hình khi thực nghiệm trên jetson nano và trong thực tế.

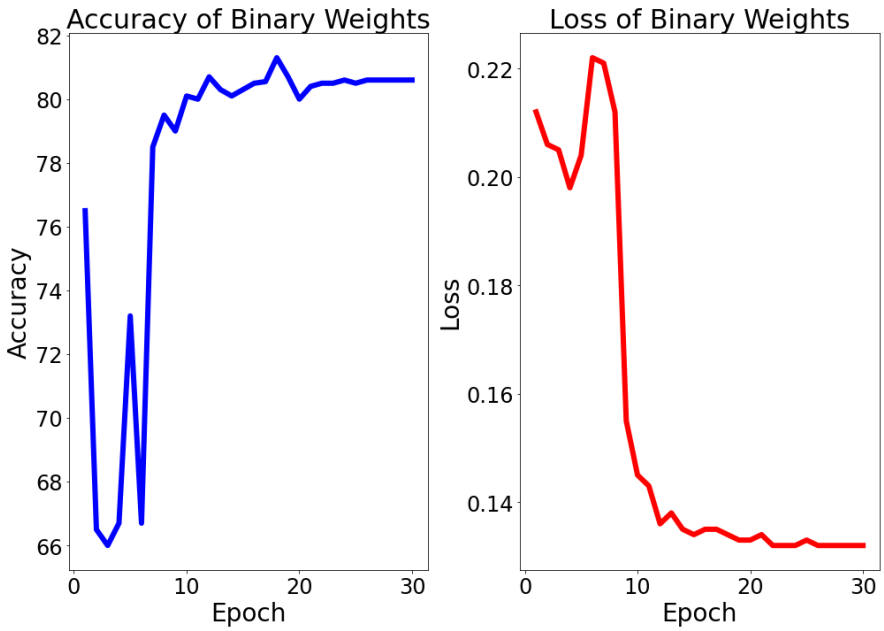
4.2 ĐÁNH GIÁ MÔ HÌNH BNN

** Mô hình BNN:*

Bảng 4-1 Thông số huấn luyện mô hình BNN

Thông tin	Epoch	Ảnh huấn luyện	Ảnh đánh giá	Kích thước ảnh
Số lượng	30	60000	10000	28x28

Nhóm thực hiện đề tài đã huấn luyện mô hình BNN với 30 epoch và sử dụng tập dữ liệu MNIST của keras và tất cả ảnh đầu vào đều có kích thước 28x28. Trong quá trình huấn luyện, mô hình sẽ lưu trọng số cho ra độ chính xác cao nhất.



Hình 4-1 Biểu diễn độ chính xác và mất mát theo của mô hình BNN

Hình 4-1 biểu diễn trình huấn luyện của mô hình BNN, trong 7 epoch đầu mô hình không ổn định, độ chính xác chênh lệch nhiều và giá trị hàm mất mát cao. Hiện tượng không ổn định có thể được giải thích bởi một số yếu tố chính. Ban đầu, việc khởi tạo ngẫu nhiên của các trọng số có thể không phù hợp, dẫn đến sự dao động lớn trong quá trình huấn luyện. Thêm vào đó, việc chọn learning rate không phù hợp cũng có thể gây ra hiện tượng này nên khi giảm learning rate qua từng epoch giúp mô hình ổn định hơn. Từ epoch thứ 8 độ chính xác của mô hình tăng ổn định hơn với chênh lệch độ chính xác ở mức ± 1% và đạt sự bão hòa sau khoảng 30 epoch. Giá trị hàm mất mát cũng giảm sâu. Điều này có thể là do mô hình đã học được các đặc trưng quan trọng của dữ liệu và các trọng số đã được điều chỉnh phù hợp.

Bảng 4-2 So sánh kết quả giữa mô hình CNN và BNN

Mô hình	Độ chính xác (%)	Thời gian thực thi (s)	Kích thước (KB)
CNN	99,2	15,2	1128
BNN	87,7	0,5	3181

Kết quả bảng 4-2 là kết quả đo đạt được khi cho các mô hình dự đoán 270 ảnh. Kết quả cho thấy mô hình BNN chưa được tối ưu (chưa hiệu chỉnh kiểu dữ liệu của trọng số) vẫn có thời gian thực thi nhanh hơn mô hình CNN 30 lần nhưng về độ chính xác lại thấp hơn 11%. Trọng số dù có giá trị 1 và -1 nhưng do sử dụng mảng numpy có kiểu dữ liệu float 64-bit để xử lý tính toán nên làm kích thước mô hình lớn hơn CNN có kiểu dữ liệu float 32-bit.

Ưu điểm của BNN so với CNN:

Tốc độ và hiệu quả tính toán:

+ Sử dụng trọng số nhị phân giúp giảm thiểu yêu cầu về bộ nhớ và tính toán, từ đó tăng tốc độ huấn luyện và thực thi.
+ Các phép tính nhị phân thường nhanh hơn và tiết kiệm bộ nhớ hơn so với các phép tính sử dụng số thực.

Khả năng tổng quát hóa: Trong một số trường hợp, việc rời rạc hóa trọng số có thể giúp mô hình tránh được hiện tượng quá khớp (overfitting).

Nhược điểm của BNN so với CNN:

Độ chính xác có thể bị giảm: Việc nhị phân hóa trọng số có thể làm giảm khả năng biểu diễn của mô hình, dẫn đến giảm độ chính xác so với các mô hình sử dụng trọng số số thực.

Lãng phí bộ nhớ: Do chưa được tối ưu hóa kiểu dữ liệu làm mô hình phình to dung lượng.

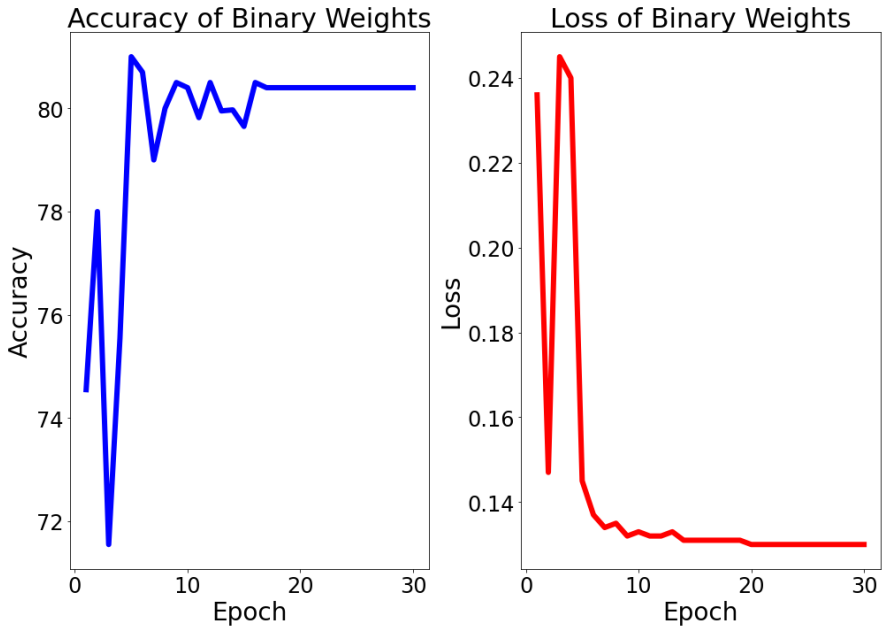
Huấn luyện phức tạp hơn: Quá trình nhị phân hóa trong quá trình huấn luyện yêu cầu các kỹ thuật đặc biệt như việc sử dụng hàm dấu hiệu (sign function) và các chiến lược huấn luyện để tối ưu hóa.

** Mô hình BNN sử dụng BitArray:*

Bảng 4-3 Thông số huấn luyện mô hình BNN+BitArray

Thông tin	Epoch	Ảnh huấn luyện	Ảnh đánh giá	Kích thước ảnh
Số lượng	30	60000	10000	28x28

Nhóm thực hiện đề tài đã huấn luyện mô hình BNN sử dụng BitArray với 30 epoch và sử dụng tập dữ liệu MNIST của keras và tất cả ảnh đầu vào đều có kích thước 28x28. Trong quá trình huấn luyện, mô hình sẽ lưu trọng số cho ra độ chính xác cao nhất.



Hình 4-2 Biểu diễn độ chính xác và mất mát theo epoch của mô hình BNN+BitArray

Hình 4-2 biểu diễn quá trình huấn luyện của mô hình BNN+BitArray, mô hình đã ổn định ở những epoch đầu tiên so với mô hình BNN và độ chính xác gần như không thay đổi. Mô hình có xu hướng nhanh đạt độ chính xác cao nhất sớm hơn và đạt trạng thái bão hòa tại epoch thứ 16, còn BNN phải huấn luyện đến epoch thứ 25.

Bảng 4-4 So sánh kết quả giữa mô hình BNN và BNN+BitArrray

Mô hình	Độ chính xác (%)	Thời gian thực thi (s)	Kích thước (KB)
CNN	99,2	15,2	1128
BNN	87,7	0,5	3181
BNN+BitArray	87,2	0,45	51

Theo như kết quả trong bảng 4-4 là kết quả đo đạt được khi cho các mô hình dự đoán 270 ảnh, mô hình BNN mới đã giảm kích thước đi hơn 60 lần so với BNN trước đó và giảm đi 32 lần so với CNN. Vì lúc này, khi lưu các trọng số, nhóm đã sử dụng thư viện BitArray để chuyển các giá trị của trọng số chỉ là 1-bit. Mô hình này có thể xem tương đối cân bằng, dù độ chính xác thấp hơn CNN nhưng thời gian thực thi và kích thước mô hình được cải thiện đáng kể.

Ưu điểm của BNN+BitArray so với BNN:

Tiết kiệm bộ nhớ: các trọng số của mô hình BNN+BitArray đều là 1-bit nên kích thước mô hình đã giảm đi đáng kể.

Thời gian thực thi: mô hình BNN+BitArray cho ra thời gian dự đoán 270 ảnh trong 0,45 giây, dù nhanh hơn chỉ 0,05 giây nhưng thể hiện việc tối ưu có hiệu quả.

Nhược điểm của BNN+BitArray so với BNN:

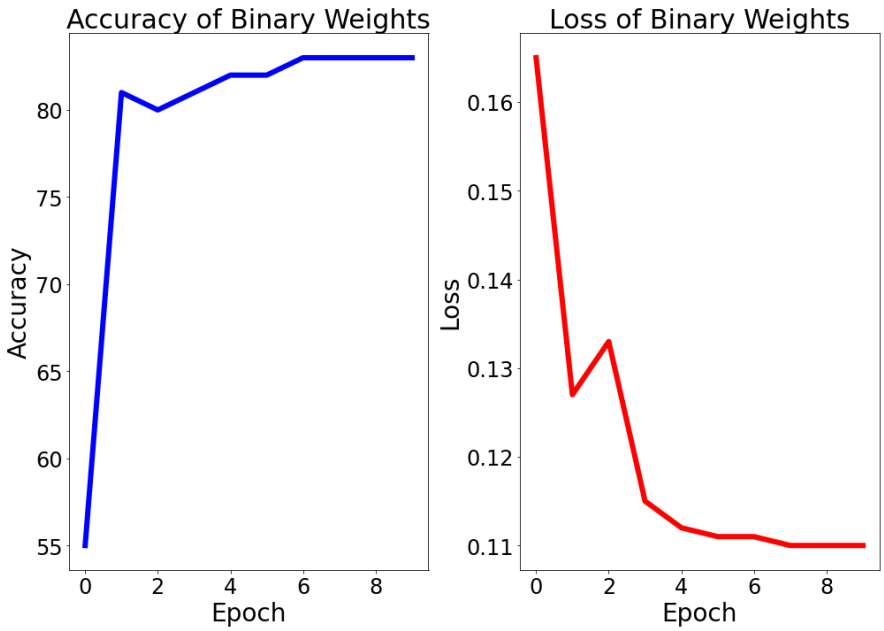
Phức tạp hơn khi huấn luyện và chạy thử nghiệm:

+ Sau khi huấn luyện, phải chuyển đổi các trọng số về dạng chuỗi 1-bit và lưu trữ kích thước của các trọng số đó.
+ Chạy thử nghiệm: Dựa vào kích thước của các trọng số trước đó, phải chuyển đổi chuỗi 1-bit của các trọng số về lại kích thước đúng của nó.

Bảng 4-5 Thông số huấn luyện mô hình BNN+K-Means

Thông tin	Epoch	Ảnh huấn luyện	Ảnh đánh giá	Kích thước ảnh
Số lượng	10	60000	10000	28x28

Nhóm thực hiện đề tài đã huấn luyện mô hình BNN với 10 epoch và sử dụng tập dữ liệu MNIST của keras và tất cả ảnh đầu vào đều có kích thước 28x28. Trong quá trình huấn luyện, mô hình sẽ lưu trọng số cho ra độ chính xác cao nhất.



Hình 4-3 Biểu diễn độ chính xác và mất mát theo epoch của mô hình BNN+K-Means

Hình 4-3 biểu diễn quá trình huấn luyện của mô hình BNN+K-Means. Với mô hình này. Mô hình này vẫn sử dụng kỹ thuật BitArray và kết hợp thêm kỹ thuật phân cụm của K-Means. Nhóm đã cho mô hình học qua từng cụm của từng nhãn từ đó đưa ra các trọng số phù hợp. Để tránh overfitting chỉ huấn luyện mô hình trong 10 epoch. Kết quả cho thấy epoch thứ hai có độ chính xác tăng vượt bậc so với epoch đầu tiên nhưng từ đó độ chính xác chỉ tăng thêm một lượng nhỏ và bắt đầu có xu hướng tuyến tính. Điều đó cho thấy việc phân cụm dữ liệu giúp dữ liệu học nhanh hơn và độ chính xác cũng tăng lên so với không phân cụm. Bên cạnh đó, hàm mất mát cũng có xu hướng ngược lại so với độ chính xác.

Bảng 4-6 So sánh kết quả giữa mô hình BNN+BitArrray và BNN+K-Means

Mô hình	Độ chính xác (%)	Thời gian thực thi (s)	Kích thước (KB)
BNN+BitArray	87,2	0,45	51
BNN+K-Means	89	0,5	51

Theo như kết quả trong bảng 4-6 là kết quả đo đạt được khi cho các mô hình dự đoán 270 ảnh. Việc áp dụng thuật toán K-Means đã tăng độ chính xác của mô hình dù vẫn giữ nguyên kích thước. Nhưng bù lại, việc phải tính toán trọng số qua từng cụm của từng sẽ làm tăng thời gian huấn luyện. Với mô hình BNN này, chúng ta có thể ứng dụng trên các bài toán phân loại trong cuộc sống đòi hỏi thời gian thực thi nhanh và kích thước mô hình nhỏ.

Ưu điểm mô hình BNN+BitArray so với BNN+K-Means:

Độ chính xác: mô hình BNN+K-Means đã cải thiện độ chính xác tăng gần 3% so với BNN+BitArray.

Nhược điểm mô hình BNN+BitArray so với BNN+K-Means:

Thời gian huấn luyện: việc cho mô hình phải học qua từng cụm của từng nhãn làm mất khá nhiều thời gian.

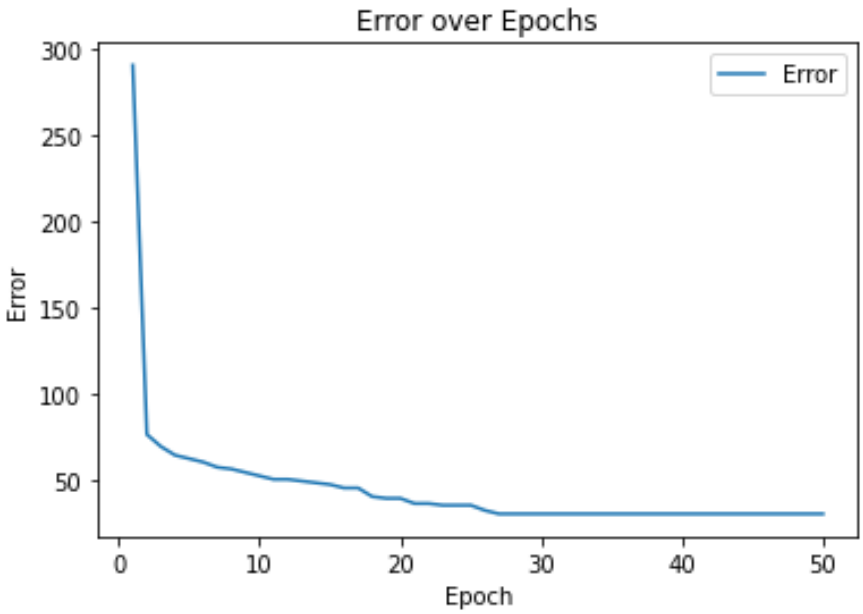
Tóm tại, mô hình mạng nơ-ron nhị phân (BNN) mang lại nhiều lợi ích về mặt hiệu quả tính toán và bộ nhớ, đồng thời đạt được độ chính xác tương đối tốt trên tập dữ liệu kiểm tra. Tuy nhiên, mô hình có thể gặp khó khăn trong việc duy trì độ chính xác cao so với các mô hình sử dụng trọng số số thực. Việc sử dụng các kỹ thuật đặc biệt trong quá trình huấn luyện và đánh giá là cần thiết để đạt được kết quả tốt nhất. Tổng quan, BNN là một giải pháp hợp lý cho các ứng dụng đòi hỏi tốc độ và hiệu quả bộ nhớ, đặc biệt là trên các thiết bị có tài nguyên hạn chế.

4.3 ĐÁNH GIÁ MÔ HÌNH LSVM KẾT HỢP VỚI K-MEANS

Bảng 4-7 Thông số huấn luyện mô hình LSVM+K-Means

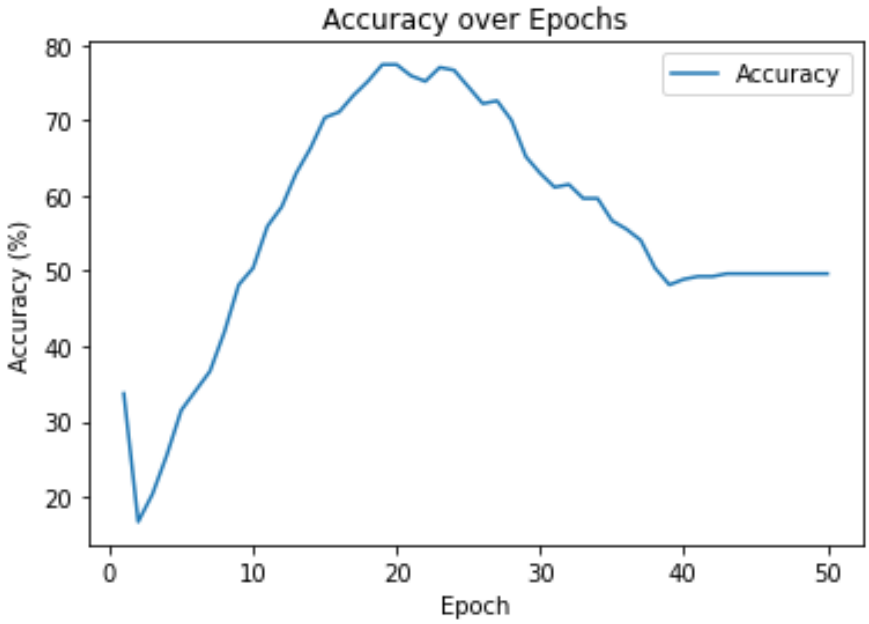
Thông tin	Epoch	Ảnh huấn luyện	Ảnh đánh giá	Kích thước ảnh
Số lượng	50	48000	10094	28x28

Nhóm thực hiện đề tài đã huấn luyện mô hình LSVM với 50 epoch và sử dụng tập dữ liệu MNIST của keras và tất cả ảnh đầu vào đều có kích thước 28x28. Trong quá trình huấn luyện, mô hình sẽ lưu thông số cho ra độ chính xác cao nhất.



Hình 4-4 Biểu diễn lỗi theo epoch của mô hình LSVM+K-Means

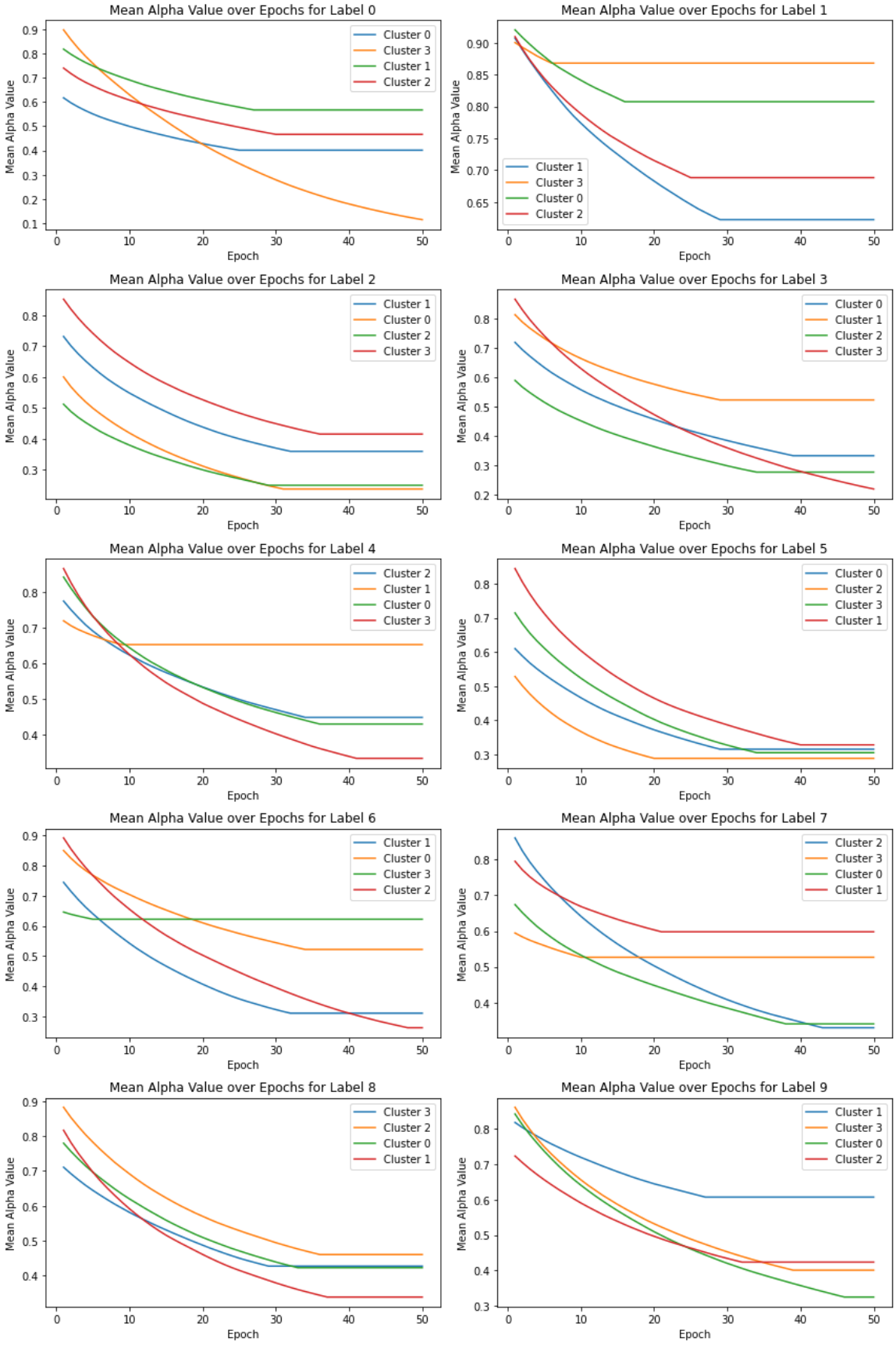
Hình 4-4 biểu diễn sự suy giảm lỗi của phép XOR popcount ảnh huấn luyện và ảnh đánh giá trong quá trình huấn luyện của mô hình LSVM sử dụng K-Means. Lỗi giảm khá nhanh vì nhanh chóng đạt giới hạn lỗi đặt ra vì nhóm thực hiện đề tài đã giới hạn lỗi tối thiểu là 34 lỗi vì nếu không có lỗi nào thì chỉ có hình hoàn toàn đen hoặc hoàn toàn trắng. Điều này sẽ làm mất khả năng phân loại ảnh của mô hình.



Hình 4-5 Biểu diễn độ chính xác theo epoch của mô hình LSVM+K-Means

Hình 4-5 biểu diễn độ chính xác của mô hình LSVM sử dụng thuật toán phân cụm K-Means qua từng epoch. Tại mỗi cụm của tập huấn luyện, mô hình sẽ tính toán và đưa ra một ma trận ảnh đại diện và ngưỡng riêng của cụm đó dùng để nhị phân hóa. Sau đó, các ma trận đại diện được XOR popcount với từng cụm trong tập đánh giá từ đó mô hình sẽ hiệu chỉnh ngưỡng nhị phân như hình 4-6 cho đến khi lỗi đạt giới hạn nhỏ nhất và mô hình sẽ lưu ma trận đại diện và mức ngưỡng cho ra độ chính xác tốt nhất.

Giá trị ngưỡng sẽ được điều chỉnh theo quy tắc: nếu lỗi trung bình lớn hơn mục tiêu lỗi và giá trị ngưỡng hiện tại nhỏ hơn hoặc bằng 0, ngưỡng sẽ được tăng lên một lượng nhỏ tỉ lệ với lỗi trung bình. Nếu lỗi trung bình lớn hơn mục tiêu lỗi và giá trị ngưỡng hiện tại lớn hơn 0, ngưỡng sẽ được giảm xuống một lượng nhỏ tỉ lệ với lỗi trung bình. Nếu lỗi trung bình đạt mục tiêu, giá trị ngưỡng sẽ không thay đổi. Cuối cùng, giá trị ngưỡng được tính toán và lưu trữ để theo dõi sự thay đổi của nó qua từng epoch, đảm bảo rằng giá trị ngưỡng được điều chỉnh liên tục. Tuy nhiên, độ chính xác của mô hình có xu hướng suy giảm sau epoch thứ 20, do việc sử dụng thuật toán giảm ngưỡng qua từng epoch. Điều này cho thấy rằng việc điều chỉnh giá trị ngưỡng qua các epoch có thể cần được tinh chỉnh thêm để tránh làm giảm hiệu quả của mô hình.



Hình 4-6 Biểu diễn giá trị ngưỡng theo epoch của mô hình LSVM+K-Means

Trong hình 4-6, tương ứng với mỗi cụm của từng nhãn sẽ có một ngưỡng khác nhau. Khi huấn luyện, mô hình sẽ được giảm ngưỡng qua từng epoch để đánh giá độ chính xác của mô hình. Mô hình đạt độ chính xác cao nhất tại epoch số 20, do cách thức cập nhật ngưỡng có thể làm tăng hoặc giảm độ chính xác một cách ngẫu nhiên nên nhóm thực hiện đề tài chạy thử qua 50 epoch và lưu thông số mô hình tại epoch có độ chính xác cao nhất.

Bảng 4-8 Kết quả đạt được của mô hình LSVM+K-Means

Mô hình	Kích thước ảnh	Số lượng ảnh thực nghiệm	Độ chính xác (%)	Thời gian thực thi (s)	Kích thước mô hình (KB)
SVM	28x28	270	94	1,1	64,874
LSVM+ K-Means	28x28	270	81	0,24s	70

Theo như kết quả trong bảng 4-8 là kết quả đo đạt được khi cho các mô hình dự đoán 270 ảnh. Kích thước mô hình LSVM+K-Means nhỏ hơn mô hình SVM 32-bit khá nhiều nhưng vẫn lớn hơn mô hình BNN+K-Means một ít, do nó phải lưu thêm ngưỡng của mỗi cụm để nhị hóa ảnh khi chạy thử nghiệm. Thời gian thực thi của mô hình cũng rất nhanh và nhanh nhất trong các mô hình trong đề tài này nhưng độ chính xác chỉ dừng lại mức tạm chấp nhận được.

Ưu điểm của mô hình LSVM+K-Means so với SVM:

Thời gian thực thi rất nhanh: thời gian thực thi 270 tấm ảnh nhanh hơn 4,5 lần so với SVM.

Kích thước nhỏ: so với mô hình SVM 32-bit thì mô hình này có kích thước nhỏ hơn rất nhiều.

Nhược điểm của mô hình LSVM+K-Means so với SVM:

Quá trình huấn luyện: đòi hỏi việc huấn luyện và hiệu chỉnh mô hình nhiều lần để cho kết quả tối ưu.

Kích thước mô hình phụ thuộc vào tập dữ liệu: nếu tập dữ liệu có nhiều đặc trưng và phải phân cụm nhiều hơn thì đòi hỏi mô hình này phải lưu nhiều thông tin của mỗi cụm, làm cho kích thước mô hình có thể lớn hơn.

Tóm lại, mô hình LSVM+K-Means cho kết quả phù hợp khi yêu cầu một mô hình thực thi nhanh. Mô hình này sẽ phù hợp với các ứng dụng tiền xử lý hệ thống, không cần phân loại quá chính xác và áp dụng trên các thiết bị nhỏ tiêu thụ năng lượng thấp, tiết kiệm tài nguyên hệ thống.

4.4 SO SÁNH KẾT QUẢ GIỮA CÁC MÔ HÌNH

Bảng 4-9 So sánh kết quả của các mô hình

Mô hình	Độ chính xác (%)	Thời gian thực thi (s)	Kích thước (KB)
CNN	99,2	15,2	1128
BNN	87,7	0,5	3181
BNN+Bitarray	87,2	0,45	51
BNN+K-Means	89	0,5	51
SVM	94	1,1	64,874
LSVM+K-Means	81	0,24	70

Nhận xét:

+ Mô hình CNN đạt được độ chính xác cao nhất (99.2%) nhưng có thời gian thực thi lâu nhất (15.2 giây) và kích thước mô hình ở mức khá tốt (1128KB) do mô hình thuộc 32-bit việc tính toàn và lưu trữ cũng sẽ hao tốn tài nguyên hơn.
+ Mô hình BNN ban đầu đạt được độ chính xác ở mức tốt (87,7%) và rút ngắn thời gian xử lý ít hơn 30 lần so với mô hình CNN nhưng chưa được tối ưu nên kích thước của mô hình còn khá lớn.
+ Mô hình BNN+BitArray là phiên bản tối ưu hơn của BNN, mô hình đã áp dụng kỹ thuật lưu mô hình 1-bit nên đã giảm kích thước ít hơn

62 lần (51KB) so mới mô hình BNN ban đầu. Bên cạnh đó thời gian thực thi đã giảm đi một its (0,45 giây), về độ chính xác cũng giảm đi một ít (87,2%).

+ Mô hình BNN+K-Means là phiên bản tốt nhất mà nhóm đạt được. Vẫn giữ được kích thước nhỏ 1-bit nhưng độ chính xác đạt được lên đến 89%. Thời gian thực thi hầu như không thay đổi với mô hình BNN.
+ Mô hình SVM của scikit-learn đã áp dụng kỹ thuật kernel trick để tối ưu các siêu mặt rất tốt. Độ chính xác của mô hình lên đến 94% nhưng do thuật toán khá phức tạp nên thời gian thực thi mất 1,1s. Đặc biệt là khi lưu mô hình này 32-bit thì kích thước mô hình tăng rất lớn (64MB).
+ Mô hình LSVM là phiên bản cải thiện kích thước và thời gian thực thi của SVM. Kích thước đã giảm hơn 926 lần so với mô hình SVM và thời gian thực thi cũng giảm hơn 4 lần. Dù vậy nhưng mô hình LSVM lại đạt được độ chính xác chỉ 81%.

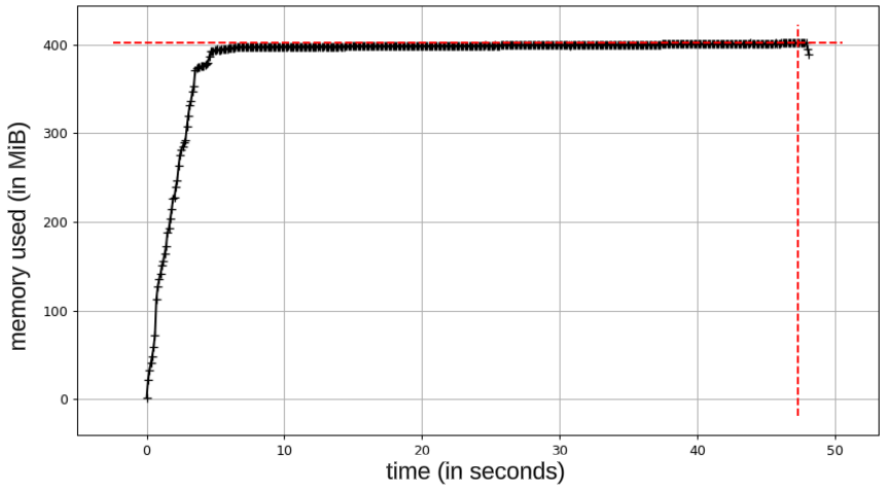
4.5 SO SÁNH TÀI NGUYÊN SỬ DỤNG GIỮA CÁC MÔ HÌNH

Bảng 4-10 Sử dụng tài nguyên bộ nhớ của các mô hình

Mô hình	Bộ nhớ sử dụng (MB)
CNN	400
BNN	350
BNN+BitArray	350
BNN+K-Means	350
SVM	225
LSVM+K-Means	340

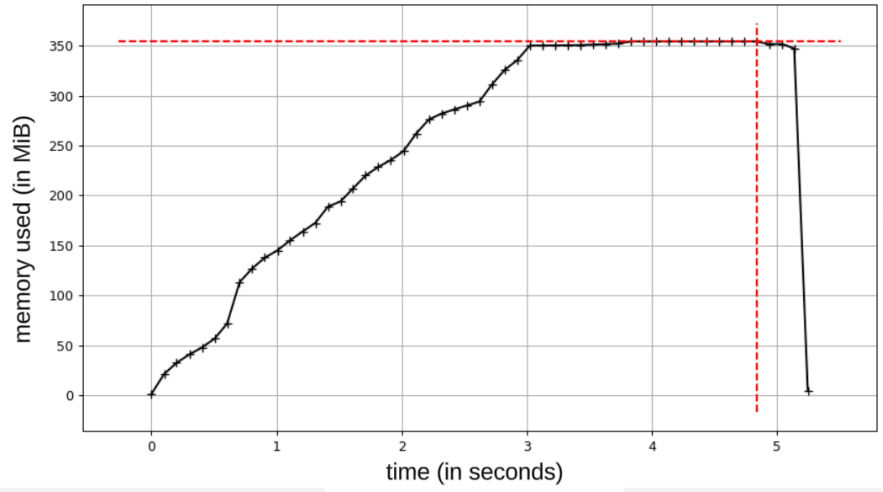
Bảng 4-10 là kết quả sử dụng bộ nhớ của mô hình. Kết quả cho thấy, mô hình CNN sử dụng bộ nhớ nhiều nhất vì nó là mô hình đòi hỏi phép toán số thực. Các mô hình biến thể của BNN và LSVM+K-Means sử dụng bộ nhớ ít hơn CNN

do các phép toán nhị phân nên các mô hình tiêu tốn bộ nhớ tương đối giống nhau. Mô hình SVM sử dụng bộ nhớ ít nhất do được hỗ trợ mạnh mẽ từ thư viện scikit-learn.



Hình 4-7 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình CNN

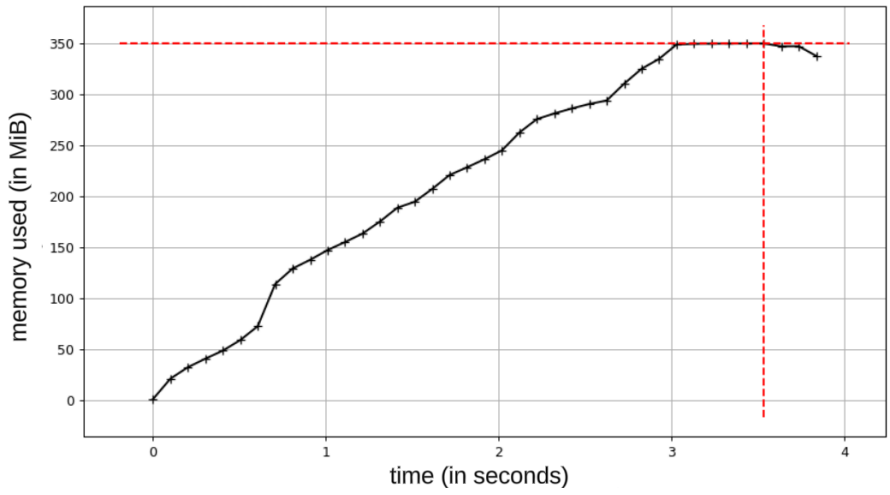
Theo hình 4-7, vừa bắt đầu mô hình CNN đã nhanh chóng tăng lượng bộ nhớ sử dụng – 400MB và sử dụng cố định lượng bộ nhớ đó suốt quá trình mà mô hình chạy thực nghiệm. Khi dự đoán, mô hình cần phải tính toán xác suất và sử dụng các phép toán số thực nên cần nhiều dung lượng bộ nhớ.



Hình 4-8 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN

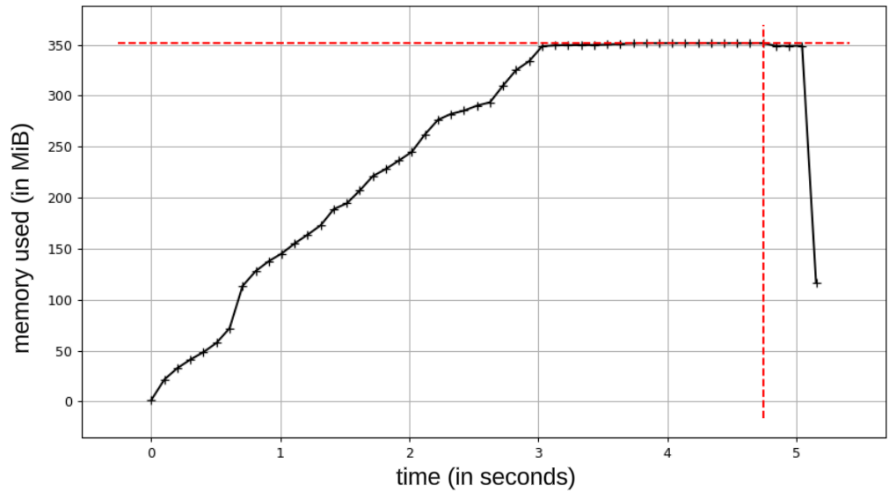
Hình 4-8 cho chúng ta thấy mô hình BNN vừa sử dụng bộ nhớ ít hơn CNN mà quá trình tăng dung lượng bộ nhớ sử dụng cũng khác đi. Đồ thị quá trình sử dụng tăng tương đuối tuyến tính góc 45 độ, điều này giúp bớt sự phức tạp của mô

hình khi xử lý, có thể tận dụng không gian trống của bộ nhớ cho những tác vụ khác.



Hình 4-9 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN+BitArray

Mô hình BNN+BitArray sử dụng bộ nhớ tối ưu hơn mô hình BNN. Trong hình 4-9, mô hình BNN+BitArray tăng lượng bộ nhớ sử dụng chậm hơn và thời gian sử dụng bộ nhớ cũng ngắn hơn so với BNN. Mặc dù, khi đỉnh điểm cả hai mô hình đều sử dụng bộ nhớ giống nhau nhưng BNN+BitArray diễn ra trong khoảng 0,7 giây, còn BNN lại diễn ra suốt 2 giây.

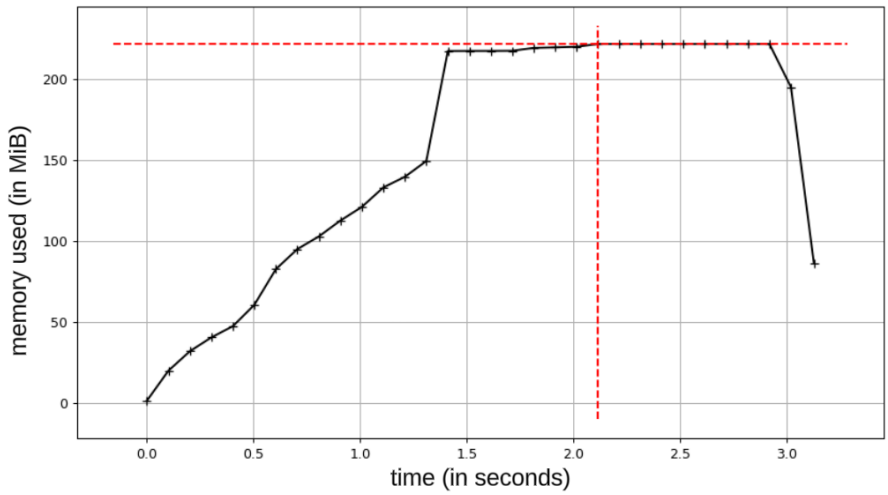


Hình 4-10 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình BNN+K-Means

Theo hình 4-10, dung lượng bộ nhớ sử dụng của mô hình BNN+K-Means khá tương đồng với BNN. Về cả quá trình tăng bộ nhớ sử dụng đến thời gian sử

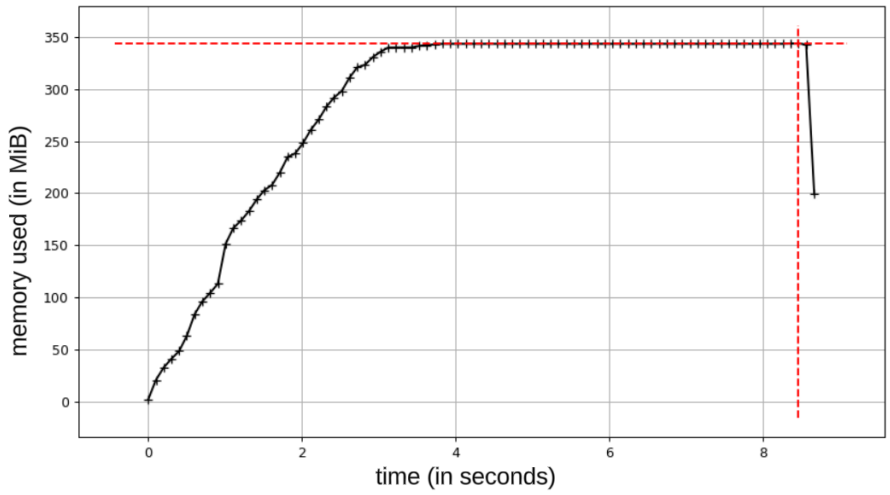
dụng bộ nhớ đỉnh điểm đều khá giống nhau. Nhưng cùng một tác vụ, sử dụng mô hình BNN+K-Means vẫn tốt hơn so với BNN do độ chính xác của BNN+K-Means cao hơn.

Tóm lại, các mô hình BNN, BNN+BitArray và BNN+K-Means có kết quả sử dụng bộ nhớ khá giống nhau, tăng tuyến tính trong một nửa thời gian chạy thực nghiệm và đạt đỉnh (350MB) ở nửa thời gian chạy còn lại. Mô hình BNN+BitArray là mô hình 1-bit và không chứa các ma trận cụm đại diện như BNN+K-Means nên mô hình chiếm dụng bộ nhớ tốt hơn 2 mô hình còn lại.



Hình 4-11 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình SVM

Hình 4-11 cho chúng ta thấy mô hình SVM đã được tối ưu bởi thư viện scikit-learn nên đã tối ưu rất mạnh mẽ khi chạy, mô hình sử dụng tối đa chỉ 250MB ít hơn các mô hình còn lại 100MB và thời gian đo cũng nhanh hơn các mô hình còn lại. Mô hình này sẽ phù hợp cho các thiết bị phần cứng hạn chế nhưng có khả năng mở rộng lưu trữ để có thể chứ kích thước đáng kể của mô hình.



Hình 4-12 Biểu diễn dung lượng bộ nhớ sử dụng theo thời gian của mô hình LSVM+K-Means

Trong hình 4-12, mô hình LSVM+K-Means ban đầu sử dụng bộ nhớ tăng tuyến tính nhưng khoảng thời gian sử dụng bộ nhớ đỉnh điểm lại nhiều hơn. Nhìn chung, mô hình LSVM+K-Means sử dụng bộ nhớ tương đối giống các mô hình của BNN. Tuy chưa được tối ưu như mô hình SVM nhưng mô hình này đạt kết quả tương đối ổn so với các mô hình BNN phía trên.

4.6 ĐÁNH GIÁ CÁC MÔ HÌNH TRÊN JETSON NANO

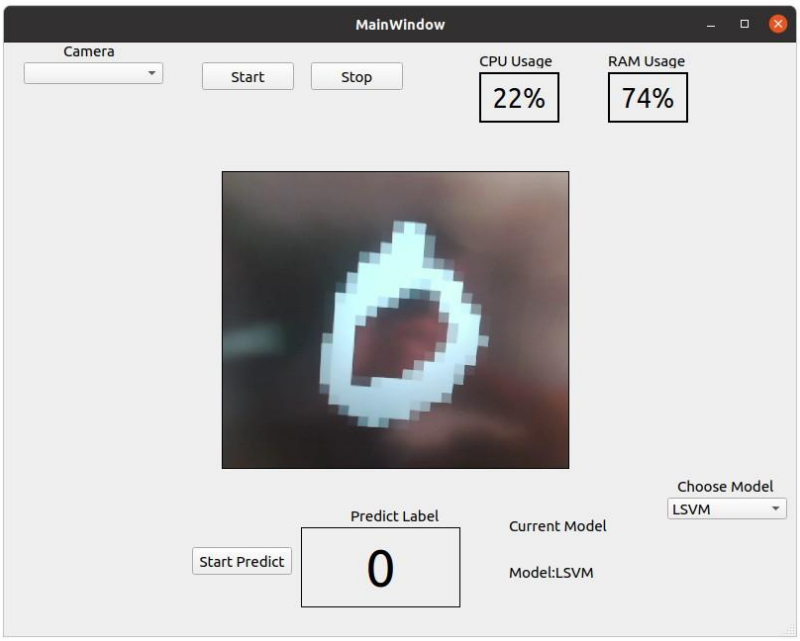
Bảng 4-11 So sánh độ chính xác và thời gian thực thi của các mô hình trên Jetson Nano

Mô hình	Độ chính xác (%)	Thời gian thực thi (s)
CNN	99,5	72
BNN	87,7	1,9
BNN + BitArray	87,1	1,9
BNN + K-Means	89	1,8
SVM	94	4,6
LSVM+K-Means	81	1,5

Kết quả của bảng 4-11 cho chúng ta thấy hiệu quả rất tốt về thời gian thực thi của các mô hình giảm kích thước. CNN cho độ chính xác gần như tuyệt đối

nhưng lại mất rất nhiều thời gian để có thể dự đoán hết 270 ảnh. Trong khi đó, mô hình giảm kích thước BNN chỉ cần 1,9 giây, nhanh hơn 37 lần so với CNN. Dù độ chính xác đi 10% nhưng đây là con số có thể chấp nhận được với những ứng dụng cần tốc độ thực thi hơn là độ chính xác. Về kết quả của LSVM+K-Means so với SVM, thời gian thực thi đã giảm hơn 3 lần nhưng độ chính xác lại suy giảm 13%.

4.7 GIAO DIỆN FRAMEWORK



Hình 4-13 Giao diện của FrameWork

Hình 4-13 là giao diện của chạy thử nghiệm mô hình kết hợp cơ chế FSM. Giao diện bao gồm các chức năng cơ bản:

+ Lựa chọn camera: giao diện cung cấp một combobox cho phép hiển thị các camera được phát hiện để người dùng lựa chọn.

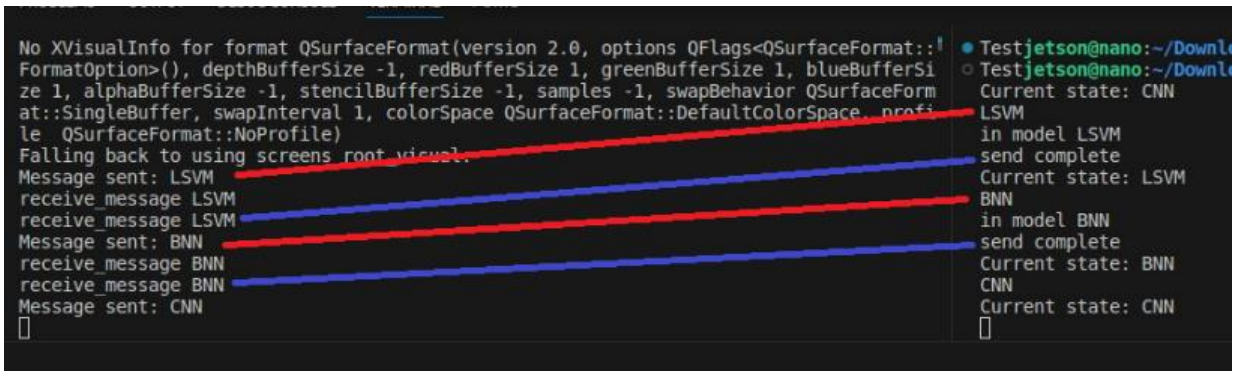
+ Khởi động camera: nút start cho phép người dùng mở camera và hình ảnh sẽ hiển thị trên khung ở giữa màn hình.

+ Tắt camera: nút stop giúp người dùng đóng camera ngay lập tức, tránh lãng phí năng lượng không cần thiết.

+ Hiển thị CPU sử dụng và RAM sử dụng: giúp người dùng xem được hệ thống đã sử dụng hết bao nhiêu tài nguyên, tránh bị quá tải hệ thống.

+ Chọn mô hình dự đoán và hiển thị mô hình đang dự đoán hiện tại: người dùng sẽ chọn mô hình muốn sử dụng, sau đó ứng dụng sẽ gửi tin nhắn đến FSM để yêu cầu chuyển mô hình, khi FSM nhận được tin nhắn, nếu thỏa mãn điều kiện thì FSM sẽ phản hồi một tin nhắn và hiển thị trong khung mô hình hiện tại.

+ Khởi động dự đoán: cho phép mô hình bắt đầu dự đoán ảnh từ camera và hiển thị nhãn dự đoán tại khung predict label.



Hình 4-14 Quá trình chuyển đổi trạng thái trong FSM

Hình 4-14 hiển thị quá trình chuyển trạng thái của ứng dụng. Bên phải là terminal của ứng dụng và bên trái là terminal của FSM. Hiện tại ứng dụng này có ba trạng thái là CNN, BNN và LSVM.

Khi khởi động, trạng thái mặc định của FSM là CNN. Khi người dùng muốn đổi sang trạng thái LSVM thì ứng dụng sẽ gửi thông điệp qua message_queue với nội dung "LSVM" sang FSM. Khi FSM nhận được nó sẽ kiểm tra xem thông điệp có hợp lệ không và xem xét điều kiện. Nếu thỏa mãn nó sẽ chuyển sang trạng thái LSVM và gửi thông điệp cho phép chuyển đổi trạng thái đến ứng dụng. Lúc này ứng dụng nhận được thông điệp trạng thái LSVM và thực hiện hành động của trạng thái là gọi mô hình LSVM để dự đoán nhãn. Cuối cùng, FSM sẽ cập nhật trạng thái hiện tại của chính mình và sẵn sàng nhận thông điệp kế tiếp.

CHƯƠNG 5

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 KẾT LUẬN

5.1.1 Kết luận về mô hình

Dự án đã nghiên cứu và triển khai được các mô hình sau: BNN, BNN sử dụng BitArray, BNN kết hợp với K-Means và LSVM kết hợp với K-Means. Kết quả thu được từ các mô hình đã được đánh giá với hai mô hình CNN và SVM dựa trên độ chính xác, tốc độ xử lý và khả năng áp dụng vào các bài toán cụ thể.

Mô hình CNN (Convolutional Neural Network) đạt độ chính xác cao trong nhận diện hình ảnh nhờ khả năng trích xuất đặc trưng mạnh mẽ từ các tầng tích chập. Trong thử nghiệm với bộ dữ liệu MNIST, CNN đạt độ chính xác hơn 99%, thể hiện khả năng phân loại hình ảnh vượt trội. Tuy nhiên, mô hình yêu cầu tài nguyên tính toán lớn và thời gian thực thi chậm, đặc biệt là khi cần GPU để huấn luyện và triển khai. Trên Jetson Nano, CNN mất 72 giây để xử lý 270 ảnh, còn với phần cứng mạnh hơn thì mất 15,2 giây. Mô hình cũng sử dụng khoảng 400MB bộ nhớ khi thực thi. Do đó, mô hình CNN phù hợp cho các ứng dụng ưu tiên độ chính xác hơn thời gian thực thi như dự báo tài chính và phân tích ảnh y khoa.

Mô hình BNN (Binary Neural Network) là một loại mạng nơ-ron với các hàm kích hoạt và trọng số là các giá trị nhị phân trong tất cả các lớp ẩn, làm giảm đáng kể yêu cầu về khả năng tính toán. Đặc biệt trên Jetson Nano, thời gian thực thi của BNN chỉ mất 1,9 cho 270 ảnh, nhanh hơn CNN hơn 37 lần. Độ chính xác đạt được 87,7% trên tập dữ liệu MNIST, thấp hơn CNN hơn 11%. Mô hình đã sử dụng khoảng 350MB bộ nhớ khi thực thi. Điều này có thể chấp nhận được trên các ứng dụng yêu cầu xử lý thời gian thực. Do chưa được tối ưu hóa kiểu dữ nên kích thước mô hình lên đến 3181KB, lớn hơn cả mô hình CNN.

Mô hình BNN kết hợp BitArray là mô hình sử dụng kỹ thuật BitArray để nén các giá trị trọng số về 1-bit để lưu trữ nhằm tiết kiệm bộ nhớ và tăng tốc độ tính toán. Thời gian thực thi là 0.45 giây cho 270 ảnh, nhanh hơn mô hình BNN 0,05 giây. Bên cạnh đó, độ chính xác của mô hình đạt 87,2%, thấp hơn so với BNN 0,05%, nhưng sự chênh lệch về thời gian thực thi và độ chính xác không đáng kể. Quan trọng lúc này kích thước mô hình giảm đáng kể xuống còn 51KB, tiết kiệm bộ nhớ hơn 60 lần so với BNN và 32 lần so với CNN và dung lượng bộ nhớ sử dụng vẫn không thay đổi so với BNN.

Mô hình BNN kết hợp với K-Means là mô hình kỹ thuật phân cụm của K-Means với BNN để tăng cường khả năng học và phân loại mô hình và vẫn sử dụng BitArray để tối ưu hóa kích thước mô hình. Kết quả đã chứng minh sự hiểu quả cho sự kết hợp này, mô hình có độ chính xác 89%, cao hơn so với BNN kết hợp với BitArray. Thời gian thực thi là 0,5 giây cho 270 ảnh, tương đương BNN. Dung lượng bộ nhớ sử dụng vẫn giữ nguyên so với hai mô hình BNN trước đó. Kích thước mô hình tương đương với BNN kết hợp với BitArray là 51KB. Từ những kết quả đạt được, mô hình này là phiên bản cân bằng về độ chính xác, thời gian thực thị và kích thước mô hình. Phù hợp cho việc triển khai trên thiết bị nhúng.

Mô hình LSVM (Linear Support Vector Machine) kết hợp với K-Means để phân cụm và tăng số lượng mẫu dữ liệu trước khi áp dụng LSVM, giúp giảm thời gian thực thi và cải thiện độ chính xác so với việc chỉ dùng SVM. Mô hình đạt độ chính xác khoảng 81% trên bộ dữ liệu MNIST. Tuy không đạt được độ chính xác chính xác cao như mô hình SVM là 94% nhưng đây là mô hình có thời gian thực thi nhanh nhất, mô hình chỉ mất 0,24 giây cho 270 ảnh. Trên Jetson Nano, mô hình cũng cho ra kết quả tương tự, điều nhanh hơn 4 lần so với SVM. Ưu điểm của mô hình tốc độ xử lý nhanh hơn cả CNN và BNN, thích hợp cho các bài toán với dữ liệu lớn và yêu cầu thời gian phản hồi nhanh. Tuy nhiên, độ chính xác của mô hình lại thấp hơn CNN và BNN và yêu cầu sự kết hợp phức tạp giữa hai thuật toán. Bên cạnh đó, mô hình sử dụng bộ nhớ cũng thấp hơn so với BNN, BNN+BitArray, BNN+K-Means là 340MB nhưng vẫn chưa tối ưu bằng mô hình

SVM của scikit-learn chỉ 225MB .Mô hình này có thể ứng dụng trong các ứng dụng thời gian thực như tìm kiếm trực tuyến, xử lý dữ liệu từ các thiết bị IoT.

Nhìn chung, dự án đã thành công trong việc phát triển và triển khai các mô hình với các kết quả cụ thể. CNN đã chứng tỏ là mô hình hiệu quả nhất cho các bài toán nhận diện hình ảnh với độ chính xác cao. Các biến thể BNN và LSVM kết hợp với K-Means cung cấp các giải pháp thay thế nhẹ hơn, thích hợp cho các ứng dụng trên thiết bị có tài nguyên hạn chế mà vẫn đạt được mức độ chính xác chấp nhận được.

5.1.2 Kết luận về framework

Framework hiện tại có thể xử lý 1 ảnh trong khoảng 350ms và chuyển đổi trạng thái nhanh và hiệu quả, đáp ứng kịp thời nhu cầu của người dùng. Framework quản lý trạng thái của các mô hình dự đoán, giúp việc chuyển đổi giữa các mô hình diễn ra mượt mà và chính xác. Điều này không chỉ đảm bảo hiệu suất mà còn tăng tính linh hoạt trong việc sử dụng các mô hình khác nhau tùy theo yêu cầu cụ thể của người dùng.

Ngoài ra, giao diện thân thiện với người dùng, cung cấp các chức năng thiết yếu như lựa chọn camera, khởi động/tắt camera, và hiển thị tài nguyên hệ thống sử dụng, giúp người dùng dễ dàng theo dõi và kiểm soát ứng dụng. Việc tích hợp khả năng hiển thị thông tin về CPU và RAM cũng giúp người dùng có thể tối ưu hóa việc sử dụng tài nguyên, tránh tình trạng quá tải.

Với cơ chế chuyển trạng thái được thiết kế thông minh và hiệu quả, framework này không chỉ tối ưu hóa việc xử lý hình ảnh mà còn đảm bảo tính nhất quán và ổn định trong quá trình vận hành. Khả năng gửi và nhận thông điệp qua message_queue giữa ứng dụng và FSM giúp đảm bảo rằng các yêu cầu của người dùng được xử lý một cách nhanh chóng và chính xác.

Tóm lại, framework kết hợp ứng dụng và cơ chế FSM đã chứng minh được hiệu quả trong việc quản lý và chuyển đổi trạng thái của các mô hình dự đoán, đồng thời mang lại trải nghiệm mượt mà và hiệu suất cao cho người dùng. Điều này mở ra tiềm năng lớn cho việc áp dụng trong nhiều lĩnh vực khác nhau, nơi việc xử lý hình ảnh nhanh chóng và chính xác là một yêu cầu quan trọng.

5.2 HẠN CHẾ

5.2.1 Hạn chế về mô hình

Mô hình CNN (Convolutional Neural Network) thường yêu cầu tài nguyên xử lý cao và dung lượng lưu trữ lớn, điều này có thể là một thách thức đối với các thiết bị có giới hạn tài nguyên như các thiết bị di động hoặc các hệ thống nhúng. Bên cạnh đó, thời gian huấn luyện và thời gian thực thi mô hình CNN có thể rất lâu, đặc biệt là với các tập dữ liệu lớn, điều này có thể hạn chế khả năng cập nhật mô hình nhanh chóng.

Mô hình BNN (Binary Neural Network) và các phiên bản BNN+BitArray, BNN+K-Means mặc dù có thời gian thực thi nhanh và dung lượng mô hình nhỏ nhưng độ chính xác của thấp hơn so với CNN, điều này có thể không đủ đối với các ứng dụng yêu cầu độ chính xác cao. Các mô hình BNN có thể không phù hợp cho các tác vụ phức tạp đòi hỏi khả năng học sâu và trích xuất đặc trưng mạnh mẽ như nhận diện hình ảnh phức tạp hoặc xử lý ngôn ngữ tự nhiên nâng cao.

Mô hình LSVM+K-Means đạt độ chính xác thấp hơn so với cả CNN và BNN, điều này làm hạn chế phạm vi ứng dụng của nó trong các tình huống yêu cầu kết quả chính xác cao. Ngoài ra, mô hình có thể gặp khó khăn trong việc xử lý các dữ liệu phi tuyến tính phức tạp và không thể học được các đặc trưng phức tạp từ dữ liệu.

5.2.2 Hạn chế về framework

Hiệu suất xử lý: Thời gian xử lý một ảnh khoảng 350ms có thể vẫn còn chậm đối với một số ứng dụng yêu cầu phản hồi thời gian thực nghiêm ngặt.

Quản lý tài nguyên: Mặc dù framework cung cấp các thông tin về CPU và RAM, việc tối ưu hóa sử dụng tài nguyên để tránh tình trạng quá tải vẫn là một thách thức.

Khả năng mở rộng: Framework hiện tại có thể gặp khó khăn khi mở rộng để hỗ trợ thêm nhiều mô hình dự đoán khác nhau hoặc xử lý các tập dữ liệu lớn hơn mà không ảnh hưởng đến hiệu suất.

Giao diện người dùng: Mặc dù giao diện thân thiện, nhưng có thể cần thêm các tính năng nâng cao để đáp ứng nhu cầu của người dùng chuyên nghiệp hơn.

5.3 HƯỚNG PHÁT TRIỂN

5.3.1 Hướng phát triển mô hình

Nâng cao độ chính xác: Nghiên cứu và áp dụng các kỹ thuật mới nhất trong học sâu để cải thiện độ chính xác của các mô hình hiện tại, đặc biệt là các mô hình nhẹ như BNN+K-Means và LSVM+K-Means.

Tối ưu hóa tài nguyên: Phát triển các phương pháp tối ưu hóa mô hình để giảm thiểu tài nguyên cần thiết cho việc triển khai CNN trên các thiết bị có tài nguyên hạn chế mà không ảnh hưởng đến độ chính xác.

Đa dạng hóa mô hình: Tích hợp và thử nghiệm các loại mô hình khác như RNN, LSTM, Transformer để xử lý các tác vụ phức tạp hơn như xử lý ngôn ngữ tự nhiên hoặc chuỗi thời gian.

5.3.2 Hướng phát triển framework

Tăng tốc độ xử lý: Nâng cao hiệu suất của framework bằng cách tối ưu hóa mã nguồn, sử dụng phần cứng chuyên dụng như GPU hoặc TPU, và triển khai các thuật toán tăng tốc.

Tối ưu hóa quản lý tài nguyên: Phát triển các thuật toán quản lý tài nguyên động để tối ưu hóa sử dụng CPU và RAM, đảm bảo hiệu suất ổn định ngay cả khi xử lý khối lượng công việc lớn.

Mở rộng khả năng hỗ trợ: Mở rộng framework để hỗ trợ nhiều mô hình khác nhau và các nguồn dữ liệu đa dạng, cho phép tích hợp linh hoạt hơn với các ứng dụng khác nhau.

Cải thiện giao diện người dùng: Tích hợp thêm các tính năng nâng cao vào giao diện người dùng để đáp ứng nhu cầu của người dùng chuyên nghiệp, chẳng hạn như khả năng tùy chỉnh cao hơn, báo cáo chi tiết và các công cụ phân tích.

Khả năng tích hợp và triển khai: Phát triển các tính năng để dễ dàng tích hợp với các hệ thống khác và triển khai trên các môi trường khác nhau như đám mây, thiết bị di động, và các hệ thống nhúng.

TÀI LIỆU THAM KHẢO

[1] T. V. Hoang, “Impact of Integrated Artificial Intelligence and Internet of Things Technologies,” pp. 1-10, 2024.

[2] J. M. M. S. L. B. Georg Rutishauser, “xTern: Energy-Efficient Ternary Neural Network,” pp. 1-8, 2024.

[3] L. H. K. D. W. L. Shien Zhu, *Unified and Optimized Ternary, Binary, and Mixed-precision Neural Network Inference on the Edge*, pp. 1-26, 2022.

[4] W. D. C. L. B. Z. G. G. Yue Li, “Ternary Neural Networks With Residual Quantization,” pp. 1-9, 2021.

[5] R. G. X. L. X. B. Haotong Qin, “Binary Neural Networks: A Survey,” pp. 1-40, 2020.

[6] V. L. Hande Alemdar, “Ternary Neural Networks for Resource-Efficient AI Applications,” pp. 1-9, 2017.

[7] F. G.-L. L. R.-M. A. L. Jair Cervantes, *A comprehensive survey on support vector machine classification: Applications, challenges and trends*, 2020.

[8] IJRASET, “A Software System for A Finite State Machine,” pp. 1-11, 2022.

[9] P. Baheti, “The Essential Guide to Neural Network Architectures,” 8 July 2021. [Trực tuyến]. Available: https://www.v7labs.com/blog/neural-network-architectures-guide. [Đã truy cập 1 March 2024].

[10] Alzubaidi, “Review of deep learning: concepts, CNN,” *Journal of Big Data*, pp. 1-74, 2021.

[11] R. N. Keiron O’Shea, “An Introduction to Convolutional Neural Networks,” pp. 1-11, 2015.

[12] H. K. Hossein Gholamalinezhad, “Pooling Methods in Deep Neural Networks, a Review,” pp. 1-16, 2020.

[13] S. S. A. Chunyu Yuan, “A comprehensive review of Binary Neural Network,” pp. 1-39, 2023.

[14] V. Kecman, “Support Vector Machines – An Introduction,” pp. 1-48, 2005.

[15] J. X. Z. Y. M. L. J. R. Huibing Wang, “Research Survey on Support Vector Machine,” pp. 1-9, 2017.

[16] D. H. A. J. A. Intisar Shadeed, “Performance Evaluation of Kernels in Support Vector Machine,” pp. 1-8, 2018.

[17] X. Z. Y. P. Z. N. H. S. Daohua Yu, “Application of Statistical K-Means Algorithm for University Academic Evaluation,” pp. 1-23, 2022.

[18] M. A. Syakur, “Integration K-Means Clustering Method and Elbow,” pp. 1-7, 2018.

[19] P. J. ROUSSEEUW, “Silhouettes: a graphical aid to the interpretation and validation,” pp. 1-13, 1987.

[20] D. D. O. Danny Matthew, “Effect of Distance Metrics in Determining K-Value in KMeans Clustering Using Elbow and Silhouette Method,” pp. 1-6, 2020.

[21] ilanschnell, 2 January 2024. [Trực tuyến]. Available: https://pypi.org/project/bitarray/. [Đã truy cập 1 March 2024].

[22] f. altendky, “pypi.org,” 16 November 2022. [Trực tuyến]. Available: https://pypi.org/project/memory-profiler/. [Đã truy cập 1 March 2024].

[23] R. D. Ms S Krishnaveni, “Comparing and Evaluating the Performance of Inter Process Communication Models in Linux Environment,” pp. 1-6, 2016.

[24] IEC, IEC 61851-1 INTERNATIONAL STANDARD, iTeh STANDARD, 2017.

[25] V. Kanade, “What Is a Finite State Machine (FSM)? Meaning, Working, and Examples,” 12 September 2023. [Trực tuyến]. Available:

https://www.spiceworks.com/tech/tech-general/articles/what-is-fsm/. [Đã truy cập 1 March 2024].

[26] M. R. R. N. P.-G. D. G. N. Febby Purnama Madrin, “The evaluation of CUDA performance on the Jetson Nano board for an image binarization task,” 2021.

[27] H. V. D. S. A. S. C. V. A. K. Manjunath R Kounte, “Design and Development of Autonomous Driving Car Using NvidiaJetson Nano Developer Kit,” pp. 1-5, 2022.

[28] M. K. D. P. Juneseo Chang, “Low-Power On-Chip Implementation of Enhanced SVM,” pp. 1-20, 2022.