HỌ TÊN SINH VIÊN: PHẠM MINH HIẾU

LỚP: D19CNPM6

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

Đề tài: ÁP DỤNG MÔ HÌNH HỌC SÂU TRIỂN KHAI ỨNG DỤNG ANDROID PHÁT HIỆN VÀ KHÔI PHỤC ẢNH MỜ

Giảng viên hướng dẫn:	ThS. Đinh Xuân Trường
Sinh viên thực hiện:	Phạm Minh Hiếu
Mã sinh viên:	B19DCCN255
Lớp:	D19CNPM6
Niên khóa:	2019-2024
Hệ đào tạo:	Đại học chính quy

Hà Nội 2024

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

Đề tài: ÁP DỤNG MÔ HÌNH HỌC SÂU TRIỂN KHAI ỨNG DỤNG ANDROID PHÁT HIỆN VÀ KHÔI PHỤC ẢNH MỜ

Giảng viên hướng dẫn:	ThS. Đinh Xuân Trường
Sinh viên:	Phạm Minh Hiếu
Mã sinh viên:	B19DCCN255
Lớp:	D19CNPM6
Niên khóa:	2019-2024
Hệ đào tạo:	Đại học chính quy

Hà Nội 2024

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:** **)**

Hà Nội, ngày tháng năm 20…

Giảng viên hướng dẫn

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:** **)**

Hà Nội, ngày tháng năm 20…

Giảng viên phản biện

LỜI CẢM ƠN

Lời đầu tiên, để có được báo cáo đồ án như ngày hôm nay em xin gửi lời cảm ơn trân trọng nhất tới Thạc sỹ Đinh Xuân Trường, giảng viên bộ môn Khoa học máy tính tại Học viện Công nghệ Bưu chính Viễn thông. Suốt thời gian qua thầy đã hỗ trợ nhiệt tình cho em, sự hỗ trợ và động viên quý báu của thầy trong cả quá trình làm đồ án giúp em hoàn thiện đồ án của mình một cách đầy đủ và trọn vẹn nhất. Thầy đã dành thời gian và tâm huyết để hướng dẫn em trong việc tìm hiểu, phát triển ý tưởng, cũng như giúp đỡ em vượt qua những thách thức khó khăn. Sự tận tâm và kiến thức sâu rộng của thầy đã giúp em có được những kiến thức quý giá và trải nghiệm thực tế trong lĩnh vực xử lý ảnh.

Tiếp theo, em xin bày tỏ lòng biết ơn chân thành tới bố và mẹ vì sự động viên trong suốt quá trình em làm đồ án tốt nghiệp. Chính bố mẹ đã là người đứng sau hỗ trợ hậu phương, sát cánh bên cạnh để em yên tâm nghiên cứu và phát triển đồ án. Nhờ có bố mẹ mà suốt những ngày tháng qua em chỉ phải tập trung tận tâm nghiên cứu đồ án, bố mẹ luôn là hậu phương vững chắc nhất của em trong suốt quãng thời gian đó.

Cuối cùng, em xin cảm ơn nhà trường Học viện Công nghệ Bưu chính Viễn thông đã tạo điều kiện cho em có những trải nghiệm quý báu này. Trong suốt 4 năm đại học, nhà trường luôn trao cho em rất nhiều cơ hội để học tập, nghiên cứu và sáng tạo. Ngày nay, là báo cáo đồ án cuối cùng của em tại trường, cũng là bài học, là trải nghiệm lớn nhất của em trong suốt quãng đời sinh viên. Cảm ơn nhà trường đã trao cho em nhiều cơ hội để cải thiện bản than trong suốt quãng đời sinh viên vừa qua.

Một lần nữa, em xin cảm ơn thầy Đinh Xuân Trường, bạn bè, gia đình và nhà trường đã ủng hộ và hỗ trợ em trong hành trình này. Mặc dù đã cố gắng hoàn thành đồ án trong phạm vi cho phép nhưng cũng không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự thông cảm, nhận xét và đóng góp ý kiến từ thầy cô.

Em xin chân thành cảm ơn !

Hà Nội, ngày tháng năm 20…

Sinh viên

Phạm Minh Hiếu

MỤC LỤC

LỜI CẢM ƠN.....i
MỤC LỤC.....ii
DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT.....v
DANH MỤC HÌNH VẼ.....vi
DANH MỤC BẢNG BIỂU.....ix
LỜI MỞ ĐẦU.....x
CHƯƠNG 1. GIỚI THIỆU CHUNG.....1
1.1. Giới thiệu hiện trạng.....1
1.2. Mục tiêu và đối tượng.....2
1.3. Định hướng giải pháp.....2
1.4. Lý thuyết liên quan đến xử lý ảnh.....3
1.4.1. Tổng quan về xử lý ảnh.....3
1.4.2. Khái niệm về phần tử ảnh.....4
1.4.3. Khái niệm về ảnh màu.....5
1.4.4. Khái niệm về ảnh xám.....6
1.4.5. Khái niệm về ảnh nhị phân.....7
1.4.6. Lược đồ xám (Histogram).....7
1.4.7. Biểu diễn ảnh.....8
1.4.8. Nâng cao chất lượng ảnh.....8
1.4.9. Biến đổi ảnh.....9
1.5. Giới thiệu về Deep Learning.....9
1.5.1. Khái niệm cơ bản về Deep Learning.....9
1.5.2. Deep Learning hoạt động như thế nào.....9
1.5.3. Ưu – nhược điểm của Deep Learning.....10
1.5.4. Ứng dụng của Deep Learning.....11
1.5.5. Dữ liệu trong Deep Learning.....14
1.6. Mạng nơ – ron.....15
1.6.1. Mạng nơ – ron nhân tạo.....15
1.6.2. Kiến trúc mạng nơ – ron tích chập.....21
1.7. Chỉ số đánh giá mô hình.....27

1.7.1. Ma trận đánh giá Confusion matrix....27
1.7.2. Độ chính xác accuracy....29
1.7.3. Precision....29
1.7.4. Recall....29
1.7.5. F1 score....30
1.7.6. Structural Similarity Index Measurement (SSIM)....30
1.7.7. PSNR....31
CHƯƠNG 2. CÁC THÀNH PHẦN HỆ THỐNG....32
2.1. Tổng quan về hệ thống....32
2.1.1. Sơ đồ hoạt động hệ thống....32
2.1.2. Luồng hoạt động các chức năng hệ thống....33
2.2. Các công cụ lập trình hệ thống....34
2.2.1. Công cụ lập trình phía Client....34
2.2.2. Công cụ lập trình phía Server....37
2.3. Các mô hình và thuật toán thử nghiệm....39
2.3.1. Mô hình học máy (Machine Learning) truyền thống....39
2.3.2. Thuật toán Fourier....43
2.3.3. Mô hình học sâu MobileNet....44
2.3.4. Mô hình học sâu NAFNet....48
CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH....53
3.1. Giới thiệu chung....53
3.2. Các bước thực hiện....54
3.2.1. Thu thập dữ liệu ảnh....54
3.2.2. Xử lý ảnh và trích chọn đặc trưng....55
3.2.3. Thử nghiệm và đánh giá các mô hình....59
3.2.4. So sánh giữa các mô hình....68
3.3. Thiết kế hệ thống....69
3.3.1. Usecase tổng quan hệ thống....69
3.3.2. Phân rã biểu đồ usecase....70
3.3.3. Scenario....71
3.3.4. State Diagram....72
3.4. Xây dựng hệ thống....72
3.4.1. Xây dựng ứng dụng....72

3.4.2. Xây dựng server.....75
3.4.3. Kết quả chương trình.....76
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....82
TÀI LIỆU THAM KHẢO.....83

DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

STT	Từ viết tắt	Diễn giải
1	API	Application Programming Interface
2	CART	Classification and Regression Trees
3	CF	Confusion matrix
4	CNN	Convolution Neural Network
5	CPU	Central Processing Unit
6	DRF	Django Rest Framework
7	FC	Fully Connected
8	FN	False Negative
9	FP	False Positive
10	GD	Gradient Descent
11	GeLU	Gaussian Error Linear Unit
12	GPU	Graphics Processing Unit
13	HTTP	Hypertext Transfer Protocol
14	ID3	Iterative Dichotomiser 3
15	JSON	JavaScript Object Notation
16	JVM	Java Virtual Machine
17	KL	Biến đổi Karhunen-Loève
18	KNN	K – Nearest Neighbors
19	MLP	Multi – layer Perceptron
20	MRI	Magnetic resonance imaging
21	NN	Neural Network
22	PSNR	Peak Signal-to-Noise Ratio
23	ReLU	Rectified Linear Unit
24	RGB	Hệ màu Red – Green - Blue
25	SCA	Simplified Channel Attention
26	SOTA	State of the Art
27	SSIM	Structural Similarity Index Measurement
28	TN	True Negative
29	TP	True Positive
30	URL	Uniform Resource Locator

DANH MỤC HÌNH VẼ

Hình 1.1. Hình ảnh mờ do rung lắc....1
Hình 1.2. Hình ảnh mờ do di chuyển [1]....1
Hình 1.3. Sơ đồ tổng quát hệ thống xử lý ảnh....3
Hình 1.4. Hình ảnh minh họa độ phân giải máy tính....4
Hình 1.5. Hình ảnh minh họa hệ màu RGB....5
Hình 1.6. Hình ảnh màu RGB [2]....5
Hình 1.7. Hình ảnh minh họa ma trận biểu diễn ảnh xám....6
Hình 1.8. Ví dụ về ảnh màu chuyển sang ảnh xám....6
Hình 1.9. Ví dụ về minh họa ảnh màu - ảnh xám - ảnh nhị phân....7
Hình 1.10. Ảnh minh họa lược đồ xám với mỗi tấm ảnh xám khác nhau [3]....7
Hình 1.11. Cải thiện chất lượng ảnh bằng khử nhiễu....8
Hình 1.12. Ảnh minh họa Deep Learning....9
Hình 1.13. Ảnh minh họa hoạt động giữa Machine Learning và Deep Learning 10
Hình 1.14. Hình ảnh minh họa xe tự lái....12
Hình 1.15. Hình ảnh minh họa bài toán phân loại....12
Hình 1.16. Hình ảnh về trợ lý ảo Siri của Apple....13
Hình 1.17. Hình ảnh minh họa các nền tảng mạng xã hội....13
Hình 1.18. Hình ảnh minh họa robot thông minh trong y tế....14
Hình 1.19. Hình ảnh minh họa tập dữ liệu chữ số viết tay MNIST dataset [5]....15
Hình 1.20. Minh họa biểu diễn Perceptron dưới dạng Neural Network [6]....15
Hình 1.21. Hình ảnh minh họa nơ-ron thần kinh sinh học [6]....16
Hình 1.22. Hình ảnh minh họa kiến trúc nơ -ron nhân tạo cơ bản nhất [7]....16
Hình 1.23. Minh họa đồ thị của hàm kích hoạt Sigmoid và Tanh [8]....17
Hình 1.24. Hình ảnh minh họa đồ thị hàm kích hoạt ReLU [9]....17
Hình 1.25.Multi – layer Perceptron với 2 hidden layer....18
Hình 1.26. Hình ảnh minh họa các ký hiệu sử dụng trong mạng nơ -ron [8]....19
Hình 1.27. Ví dụ minh họa về mạng nơ – ron....19
Hình 1.28. Mô tả thứ tự tính toán của quá trình feedfoward....20
Hình 1.29. Minh họa cho quá trình backpropagation....21
Hình 1.30. Minh họa cho phép tích chập trong không gian 2 chiều....21
Hình 1.31. Minh họa tích chập cho bài toán phân loại....22
Hình 1.32. Minh họa thực hiện tích chập với bộ lọc [10]....23
Hình 1.33. Minh họa kết quả tích chập với các bộ lọc khác nhau [3]....24
Hình 1.34. Hình ảnh minh họa stride = 2....25

Hình 1.35. Hình ảnh minh họa ma trận ảnh được padding = 1.....25
Hình 1.36. Hình ảnh minh họa Max Pooling.....26
Hình 1.37. Minh họa lớp fully connected [10].....27
Hình 1.38. Minh họa unnormalized confusion matrix và confusion matrix [11].28
Hình 2.1. Hình ảnh minh họa sờ đồ hoạt động hệ thống.....32
Hình 2.2. Hình ảnh minh họa Kotlin.....34
Hình 2.3. Hình ảnh minh họa Gson chuyển đổi dữ liệu.....35
Hình 2.4. Minh họa Retrofit.....37
Hình 2.5. Minh họa Django Rest Framework.....38
Hình 2.6. Hình ảnh minh họa cloudinary.....39
Hình 2.7. Hình ảnh minh hoạt thuật toán K - Nearest Neighbors.....40
Hình 2.8. Minh họa thuật toán Naive Bayes.....41
Hình 2.9. Hình ảnh minh họa thuật toán Decision Tree.....42
Hình 2.10. Hình ảnh minh họa thuật toán Random Forest.....43
Hình 2.11. Hình ảnh minh họa kiến trúc mô hình MobileNet [13].....45
Hình 2.12. Kiến trúc tổng quát mô hình MobileNetV2 [14].....46
Hình 2.13. Kiến trúc từng khối mô hình MobileNetV2 [14].....47
Hình 2.14. Hình ảnh so sánh kiến trúc MobileNetV1 và MobileNetV2 [14].....47
Hình 2.15. So sánh giữa các khối tích chập trong mô hình MobileNet [14].....48
Hình 2.16. Kiến trúc U-Net nhiều giai đoạn của các mạng SOTA [15].....49
Hình 2.17. Kiến trúc mạng U-Net một giai đoạn NAFNet [15].....49
Hình 2.18. Kiến trúc từng khối của mạng NAFNet [15].....50
Hình 2.19. Minh họa khối Simple Gate [15].....51
Hình 2.20. Minh họa khối SCA [15].....51
Hình 3.1. Minh họa sơ đồ xây dựng hệ thống.....53
Hình 3.2. Các folder dữ liệu tương ứng với ba loại ảnh.....54
Hình 3.3. Minh họa ba loại ảnh đối với một khung cảnh.....54
Hình 3.4. Hình ảnh minh họa tập dữ liệu GoPro.....55
Hình 3.5. Minh họa ảnh chụp vật thể bị mờ do rung lắc.....56
Hình 3.6. Hình ảnh minh họa áp dụng toán tử phát hiện biên.....57
Hình 3.7. Minh họa đoạn mã trích lọc đặc trưng và lưu vào tập dữ liệu mới.....58
Hình 3.8. Minh họa sự phân bố của các điểm dữ liệu trong không gian 3 chiều. 59
Hình 3.9. Kết quả thực nghiệm với thuật toán Fourier.....61
Hình 3.10. Đoạn mã chia dữ liệu huấn luyện đối với tập dữ liệu ảnh mờ.....62
Hình 3.11. Đoạn mã chia dữ liệu huấn luyện với tập dữ liệu ảnh nét.....63
Hình 3.12. Đoạn mã chia batch size cho quá trình huấn luyện.....63

Hình 3.13. Tổng quát kiến trúc mô hình huấn luyện.....64
Hình 3.14. Sơ đồ huấn luyện.....64
Hình 3.15. Minh họa quá trình huấn luyện mô hình MobileNet.....65
Hình 3.16. Biểu đồ minh họa accuracy và loss trên tập training và validation….65
Hình 3.17. Biểu đồ minh họa loss của quá trình huấn luyện mô hình NAFNet…67
Hình 3.18. Biểu đồ minh họa chỉ số PSNR.....67
Hình 3.19. Biểu đồ minh họa chỉ số SSIM.....67
Hình 3.20. Usecase tổng quan hệ thống.....70
Hình 3.21. Usecase phát hiện ảnh mờ.....70
Hình 3.22. Usecase khôi phục ảnh mờ.....71
Hình 3.23. State Diagram.....72
Hình 3.24. Minh họa cấu trúc thư mục phía Client.....73
Hình 3.25. Object ApiConfig phía Client.....74
Hình 3.26. interface ApiService phía Client.....74
Hình 3.27. Cấu trúc thư mục phía Server.....75
Hình 3.28. Minh họa màn hình ứng dụng chụp ảnh.....76
Hình 3.29. Minh họa màn hình ứng dụng cho việc tải ảnh lên.....77
Hình 3.30. Minh họa màn hình sau khi chọn ảnh tải lên.....78
Hình 3.31. Minh họa màn hình ứng dụng dự đoán và phân vùng ảnh mờ.....79
Hình 3.32. Minh họa màn hình ứng dụng sau khi khôi phục ảnh mờ.....80
Hình 3.33. So sánh hình ảnh trước và sau khi khôi phục.....81

DANH MỤC BẢNG BIỂU

Bảng 1-1. So sánh Max Pooling và Average Pooling [10].....27
Bảng 3-1. Kết quả thực nghiệm các mô hình học máy truyền thống.....60
Bảng 3-2. So sánh các phương pháp trong tác vụ phát hiện ảnh mờ.....69

LỜI MỞ ĐẦU

Hiện nay, cuộc cách mạng công nghiệp 4.0 đã và đang cho thấy sức ảnh hưởng to lớn của mình trong cuộc sống. Hàng loạt các công nghệ mới ra đời và liên tục phát huy hiệu năng phục vụ đời sống xã hội. Khi cuộc sống con người được nâng cao, việc lưu giữ lại những hình ảnh cũng trở nên khá quen thuộc, nhất là lúc các trang mạng xã hội đang ngày càng kết nối chúng ta lại với nhau và phát triển một cách mạnh mẽ hơn. Từ đây, nhu cầu về xử lý ảnh cũng tăng cao vì đơn giản tại thời đại này, tấm ảnh không chỉ mang tính chất lưu giữ lại hình ảnh mà còn thể hiện nhiều hơn về các khía cạnh nghệ thuật, thẩm mỹ khác. Bên cạnh việc cải thiện chất lượng ảnh sao cho sống động, rực rỡ hơn thì nhiệm vụ khôi phục hình ảnh cũng rất được quan tâm.

Cụ thể hơn, khôi phục hình ảnh bị mờ do chuyển động là một nhiệm vụ được quan tâm rất nhiều trong những năm gần đây khi được ứng dụng rất nhiều trong các lĩnh vực như tài chính, giáo dục, giao thông, y tế, nghệ thuật, lịch sử. Trong thực tế, việc một tấm ảnh quan trọng bị chụp trong một hoàn cảnh không mấy thuận lợi khiến tấm ảnh đó bị mờ không khó để bắt gặp. Đã có rất nhiều trường hợp tai nạn giao thông mặc dù hình ảnh đã được ghi lại nhưng do vận tốc, va chạm diễn ra chuyển động quá nhanh khiến tấm ảnh bị mờ và làm ảnh hưởng đến kết quả điều tra. Hay bản thân em đã từng một lần chụp ảnh nhưng do tay quá rung nên phải chụp đi chụp lại rất nhiều lần vì sau mỗi lần rung tay thì tấm ảnh lại trả ra một kết quả không tốt.

Chính vì hướng tới mục tiêu cải thiện những trở ngại khó khăn trong việc chụp ảnh nên xử lý những tấm ảnh bị mờ do chuyển động đã thôi thúc em thực hiện đề tài: "**Áp dụng mô hình học sâu triển khai ứng dụng Android phát hiện và khôi phục ảnh mờ**" này.

Nội dung trình bày trong báo cáo đồ án gồm 3 chương sau:

- **CHƯƠNG 1. GIỚI THIỆU CHUNG**

  Trình bày về bối cảnh thực tế, các mục tiêu, đối tượng và công cụ xây dựng đồ án. Bên cạnh đó giới thiệu các kiến thức nền tảng cơ bản về xử lý ảnh, ứng dụng của xử lý ảnh, Deep Learning và các chỉ số đánh giá độ hiệu quả của một mô hình.

- **CHƯƠNG 2. CÁC THÀNH PHẦN HỆ THỐNG**

  Giới thiệu về sơ đồ hoạt động tổng thể của hệ thống và luồng vận hành các chức năng. Tìm hiểu các kiến thức về thuật toán và mô hình phục vụ cho việc thử nghiệm như các thuật toán học máy truyền thống, mô hình học sâu MobileNet và mô hình học sâu NAFNet

- **CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH**

  Tập trung giới thiệu, phân tích sơ đồ xây dựng chương trình và thực hiện theo các bước đã đề ra. Tiến hành thử nghiệm các thuật toán, mô hình học máy, mô hình học sâu đã giới thiệu ở chương 2 và đưa ra đánh giá kết luận. Cuối cùng là xây dựng chương trình và đưa ra các kết quả minh họa.

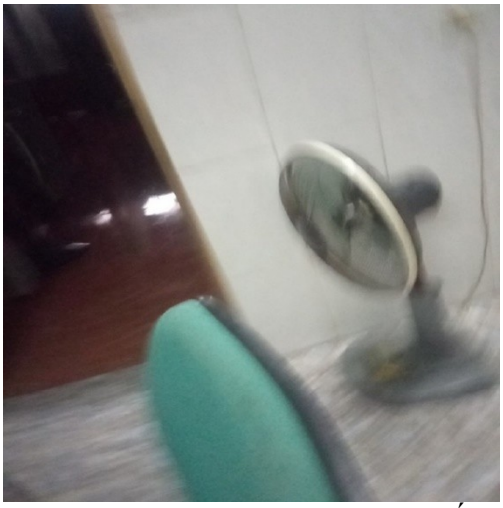
CHƯƠNG 1. GIỚI THIỆU CHUNG

Nội dung chương 1 được xây dựng để giới thiệu về những khái niệm, nền tảng kiến thức chung của đồ án. Mục đích của chương 1 để làm rõ các nội dung kiến thức liên quan sẽ được sử dụng trong đồ án. Trình tự trình bày chương 1 bao gồm các thành phần sau:

- Giới thiệu hiện trạng, mục tiêu, đối tượng và các định hướng giải pháp để xây dựng đồ án.
- Các kiến thức cơ bản liên quan đến xử lý ảnh và các ứng dụng của xử lý ảnh trong thực tế.
- Giới thiệu về Deep Learning và kiến thức về mạng nơ – ron tích chập.
- Giới thiệu các chỉ số đánh giá một mô hình hoạt động có hiệu quả hay không.

1.1. Giới thiệu hiện trạng

Hiện nay, việc bắt gặp một hình ảnh kỹ thuật số chứa các vùng bị mờ do chuyển động là khá phổ biến. Nguyên nhân tấm ảnh chứa vùng mờ đó là việc máy ảnh không thể bắt kịp chuyển động của vật thể, khiến cho tấm ảnh bị nhòe đi theo hướng chuyển động của vật thể. Nguyên nhân thứ hai có thể kể tới là việc chụp tấm ảnh trong một điều kiện camera bị rung lắc. Khi camera ở trong trạng thái không ổn định cũng gây ra hiện tượng ảnh kỹ thuật số bị nhòe đi do sự rung lắc này. Các hình ảnh minh họa dưới đây có thể phần nào giúp bạn đọc hiểu rõ hơn về một tấm ảnh kỹ thuật số bị mờ do chuyển động.



Hình 1.1. Hình ảnh mờ do rung lắc

Hình 1.2. Hình ảnh mờ do di chuyển [1]

Có thể thấy rằng, các hình ảnh trên bị mờ khiến tấm ảnh bị nhòe đi, gây mất mỹ quan và ảnh hưởng tới nhiều kết quả liên quan sau này trong nhiều lĩnh vực như tài chính, y tế, giao thông. Xuất phát từ hiện trạng thực tế đó đề tài đồ án này đã hướng tới việc xây dựng một ứng dụng cho phép phát hiện và khôi phục

ảnh mờ trên nền tảng Android. Đề tài đồ án xoay quanh việc tìm hiểu các cơ sở lý thuyết về xử lý ảnh, nhận dạng ảnh, vùng mờ và khôi phục ảnh bị mờ do chuyển động. Từ cơ sở lý thuyết đó tiến hành thử nghiệm các mô hình, thuật toán để áp dụng và xây dựng một ứng dụng cụ thể xử lý hai tác vụ chính đó là phát hiện ảnh mờ và khôi phục ảnh mờ. Hình ảnh được thử nghiệm thuộc loại hình ảnh bị mờ do chuyển động hoặc rung lắc từ camera.

1.2. Mục tiêu và đối tượng

Mục tiêu đề tài: Tìm hiểu lý thuyết và áp dụng kiến thức để xây dựng lên một ứng dụng Android giúp phát hiện, phác thảo vùng mờ và khôi phục ảnh mờ do chuyển động.

Ứng dụng sẽ cho phép người dùng chụp ảnh hoặc tải ảnh lên để nhận biết và phác thảo lại các vùng bị mờ. Sau khi nhận diện tấm ảnh bị mờ thì người dùng có thể tiếp tục khôi phục ảnh để khử mờ tấm ảnh đi giúp cho tấm ảnh đạt chất lượng tốt hơn.

Đối tượng sử dụng: Ứng dụng bao gồm tất cả mọi người có nhu cầu phát hiện độ mờ và khử mờ cho tấm ảnh của mình. Từ đây, ứng dụng giúp cải thiện chất lượng ảnh để phục vụ cho các nhiệm vụ khác trong đa dạng các lĩnh vực sau này.

1.3. Định hướng giải pháp

Theo đó, tổng quan luồng xây dựng chương trình sẽ bao gồm các bước:

- Thu thập, khảo sát dữ liệu
- Xử lý dữ liệu, trích lọc các đặc trưng cơ bản
- Thử nghiệm các thuật toán, mô hình cho các tác vụ hiện ảnh mờ và khôi phục ảnh mờ
- Đánh giá và lựa chọn mô hình phù hợp để xây dựng ứng dụng Android
- Xây dựng ứng dụng Android cho tác vụ chụp và gửi ảnh lên server
- Xây dựng server bằng Django Rest Framework để xử lý các tác vụ liên quan đến xử lý ảnh.

Ngôn ngữ xây dựng cho server là Python, xây dựng Ứng dụng Client bằng ngôn ngữ Kotlin và sử dụng Retrofit để giao tiếp giữa Client và Server

Cấu hình máy tính sử dụng để xây dựng mô hình và chương trình bao gồm:

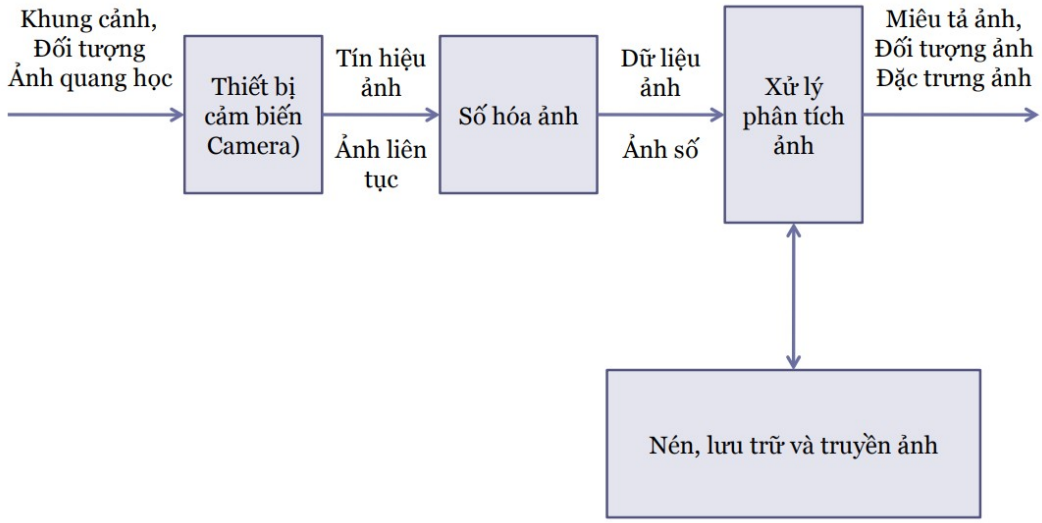
- Hệ điều hành: Window 11
- Vi xử lý: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
- Card đồ họa: NVIDIA GeForce MX250
- RAM máy: 16GB

1.4. Lý thuyết liên quan đến xử lý ảnh

1.4.1. Tổng quan về xử lý ảnh

Xử lý ảnh hay xử lý ảnh kỹ thuật số là một phân ngành trong xử lý số tín hiệu với tín hiệu xử lý là ảnh. Đây là một phân ngành khoa học mới được phát triển rất nhiều năm gần đây. Xử lý ảnh bao gồm nhiều lĩnh vực có thể kể đến như xử lý nâng cao chất lượng ảnh, nhận dạng ảnh, nén ảnh và truy vấn ảnh. Sự phát triển của xử lý ảnh đem lại rất nhiều lợi ích cho cuộc sống con người. Các ví dụ có thể kể ra liên quan đến hệ thống xử lý ảnh như hệ thống xử lý ảnh vệ tinh để phân tích không gian vũ trụ, hệ thống thăm dò địa chất, hệ thống phân tích tế bào sinh học và gần gũi với thế hệ trẻ là các phần mềm hiển thị và xử lý ảnh như Photoshop, SNOW, Inshot.

Xử lý ảnh có thể hiểu cơ bản là áp dụng thuật toán lên hình ảnh đầu vào để cho một hình ảnh đầu ra mới đã được xử lý. Đối tượng của xử lý ảnh là xử lý các ảnh tự nhiên, ảnh chụp, dữ liệu hình ảnh có nguồn gốc từ tín hiệu ảnh đặc trưng bởi biên độ và dải tần số. Các hệ thống xử lý ảnh nhìn chung sẽ thu nhận khung cảnh hoặc ảnh ở đầu vào, thực hiện các phép xử lý để tạo ra một ảnh ở đầu ra thỏa mãn các yêu cầu cảm thụ hoặc trích rút các đặc trưng của ảnh.



Hình 1.3. Sơ đồ tổng quát hệ thống xử lý ảnh

Sơ đồ khối tổng quát trên bao gồm các chức năng cơ bản sau:

- Khối thiết bị camera: Thiết bị camera thực hiện chức năng thu nhận khung hình, ảnh quang học để đưa ra tín hiệu ảnh hoặc ảnh liên tục (video)
- Khối số hóa ảnh: Đầu vào nhận các tính hiệu ảnh, ảnh liên tục để xử lý và đưa ra ảnh số, dữ liệu ảnh ở dạng các pixel.
- Khối xử lý ảnh: Đầu vào nhận dữ liệu ảnh số là các pixel để xử lý và cho kết quả đầu ra mong muốn đối với từng bài toán cụ thể.

- Khối nén, lưu trữ và truyền ảnh: Khối chức năng đảm nhiệm công việc lưu trữ và nén ảnh. Khối có thể gửi ảnh được nén cho khối xử lý ảnh hoặc nhận kết quả trực tiếp từ khối xử lý ảnh để đưa vào lưu trữ.

Xử lý ảnh có nhiều ứng dụng trong thực tế đời sống có thể kể đến như:

- Thông tin ảnh, truyền thông ảnh
- Xử lý ảnh vệ tinh, viễn thám
- Thiên văn, nghiên cứu không gian, vũ trụ
- Người máy, tự động hóa
- Máy thông minh, thị giác máy nhân tạo
- Sinh học, y học
- Giám sát kiểm soát và quân sự

1.4.2. Khái niệm về phần tử ảnh

Trong biểu diễn ảnh số, ảnh là một mảng 2 chiều gồm M hàng và N cột với f(x,y) thể hiện mức xám của ảnh tại tọa độ (x,y). Các tọa độ điểm (x,y) này được coi là các điểm ảnh hay còn được coi là các phần tử của ảnh. Phần tử ảnh là đơn vị cơ bản nhất của hình ảnh, mang cho mình thông tin vị trí và giá trị nồng độ của hình ảnh, những thông tin về màu sắc, tỉ lệ 3 màu cơ bản gồm đỏ, xanh lam và xanh lá.

Ảnh trong thực tế là một ảnh liên tục về không gian và giá trị độ sáng. Để có thể xử lý ảnh bằng máy tính cần thiết phải số hóa ảnh. Trong quá trình số hóa, tín hiệu liên tục biến đổi sang tín hiệu rời rạc thông qua quá trình lấy mẫu (rời rạc hóa về không gian) và lượng hóa thành phần giá trị. Trong quá trình này sử dụng khái niệm Pixel để biểu diễn các phần tử của bức ảnh.

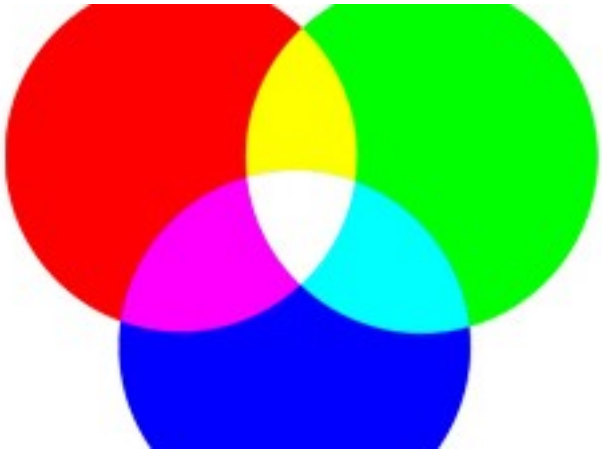
Người ta gọi chung với một khái niệm mới mang tên pixel để đại diện cho các phần tử ảnh. Khi quan sát màn hình trong chế độ đồ họa, màn hình không liên tục mà gồm nhiều điểm ảnh nhỏ, mỗi điểm ảnh này sẽ gồm một cặp tọa độ x,y và giá trị màu như đã nêu trên. Các pixel tạo lên độ phân giải cho tấm ảnh.

Hiện nay màn hình máy tính, tivi có các độ phân giải khác nhau, phổ biến nhất là màn hình HD 1280 x 720 hay Full HD 1920 x 1080.

Hình 1.4. Hình ảnh minh họa độ phân giải máy tính

1.4.3. Khái niệm về ảnh màu

Ảnh màu là một ma trận các pixel mà ở đó mỗi pixel biểu diễn một điểm màu [2]. Mỗi điểm màu được biểu diễn bằng bộ ba số (R,G,B) tương ứng là ba kênh màu đỏ (Red), xanh lá (Green) và xanh lam (blue). Ngoài ra, để hiểu rõ hơn về ba kênh màu (R,G,B) thì đây là ba màu chính của ánh sáng khi tách ra từ lăng kính. Khi trộn ba màu này theo các tỉ lệ nhất định có thể tạo ra nhiều các màu khác nhau.



Hình 1.5. Hình ảnh minh họa hệ màu RGB

Một ví dụ về tấm ảnh màu được thể hiện như hình dưới đây:

Hình 1.6. Hình ảnh màu RGB [2]

1.4.4. Khái niệm về ảnh xám

Khác với khái niệm ảnh màu, ảnh xám (gray image) hay còn gọi là ảnh đơn sắc (monochromatic). Ví dụ nếu là ảnh 8 mức xám thì mỗi điểm ảnh sẽ có giá trị nằm trong khoảng [0-7] , ảnh 256 mức xám thì mỗi điểm ảnh sẽ có giá trị nằm trong khoảng [0-255]. Ở đây, giá trị của điểm ảnh bằng 0 đại diện cho mức xám tối nhất (đen), giá trị điểm ảnh cao nhất đại diện cho điểm ảnh sáng (trắng).

Ngoài ra, ta cũng có thể biến đổi một tấm ảnh màu sang ảnh xám qua công thức tính độ sáng dưới đây:

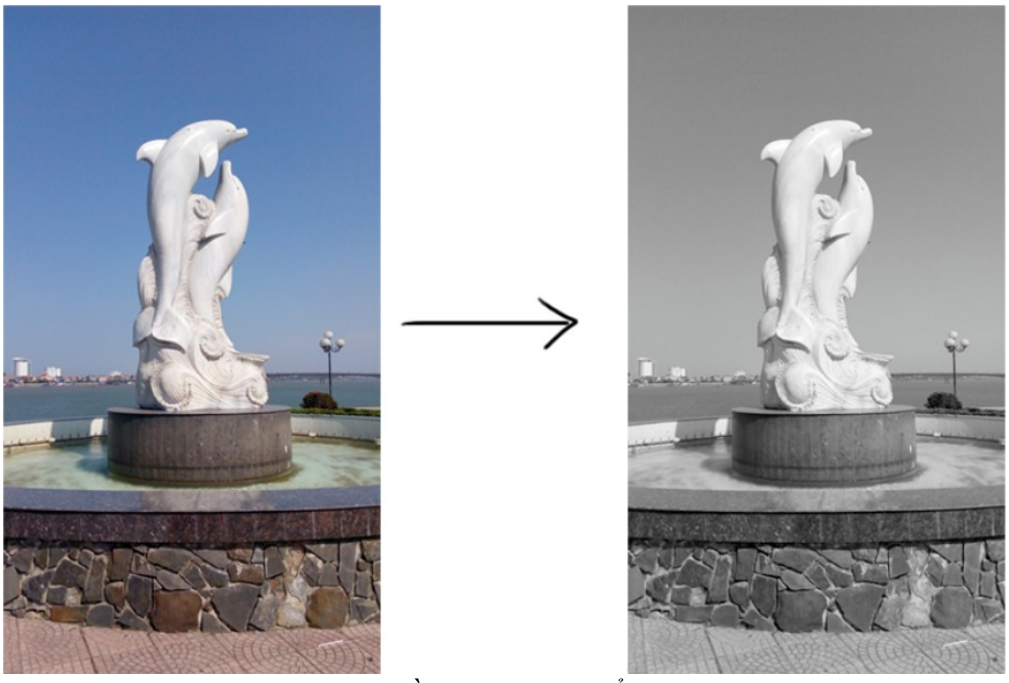
$$S = R \times 0.299 + G \times 0.587 + B \times 0.114$$

Công thức trên với S là độ sáng của điểm ảnh ở ảnh xám (gray image), R, G, B là đại diện cho mức sáng của ba kênh màu đỏ (Red), xanh lá (Green), xanh lam (Blue) của ảnh màu. Một ví dụ khác về biểu diễn ảnh xám với 256 mức xám:

0	4	16	7
154	136	0	98
255	198	154	1
200	236	234	0

Hình 1.7. Hình ảnh minh họa ma trận biểu diễn ảnh xám

Hay một ví dụ khác về chuyển đổi ảnh màu sang ảnh xám dưới đây:



Hình 1.8. Ví dụ về ảnh màu chuyển sang ảnh xám

Trong đồ án cũng sử dụng hình xám để thực hiện các công đoạn phát hiện vùng mờ cho bức ảnh qua phép biến đổi Fourier sẽ được trình bày ở phần sau.

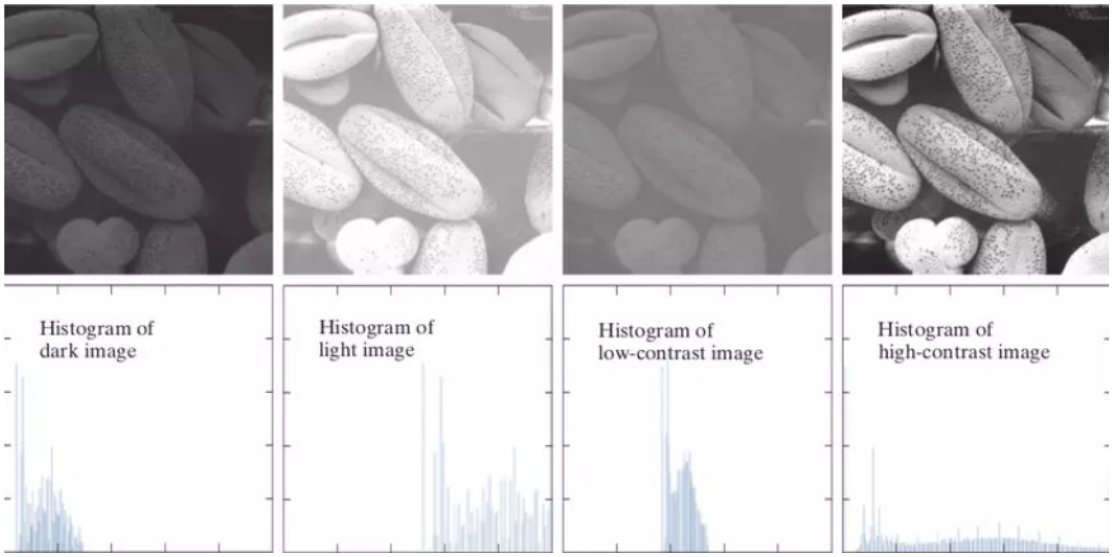
1.4.5. Khái niệm về ảnh nhị phân

Đúng như tên gọi, ảnh nhị phân chỉ chứa những pixel có giá trị 0 hoặc 1 trong đó 0 chỉ màu đen và 1 chỉ màu trắng. Hình ảnh này còn được gọi là Đơn sắc.

Hình 1.9. Ví dụ về minh họa ảnh màu - ảnh xám - ảnh nhị phân

1.4.6. Lược đồ xám (Histogram)

Lược đồ mức xám cho biết tần suất xuất hiện của mỗi điểm ảnh - pixel trong một bức ảnh ứng với mức xám tương ứng. Cụ thể, lược đồ xám (histogram) được biểu diễn bằng một hàm rời rạc $h(r_k)=n_k$, trong đó r_k là mức xám thứ k và n_k là số lượng điểm ảnh có mức xám là r_k. Thông thường histogram được tiêu chuẩn hóa bằng cách chia $h(r_k)$ cho n, với $n=\sum n_k$. Ví dụ, một ảnh đa mức xám sử dụng 8 bit, có 256 mức xám từ 0 tới 255. Lược đồ mức xám sẽ có trục hoành chạy từ 0 tới 255 và trục tung chính là tổng số điểm ảnh có mức xám tương ứng. Biểu đồ này tuy đơn giản nhưng có nhiều ứng dụng trong các bài toán giãn độ tương phản, phân ngưỡng ảnh và biến từ ảnh mức xám sang ảnh nhị phân.



Hình 1.10. Ảnh minh họa lược đồ xám với mỗi tấm ảnh xám khác nhau [3]

1.4.7. Biểu diễn ảnh

Trong biểu diễn ảnh, thường dùng các phần tử đặc trưng của ảnh là pixel. Có thể xem một hàm hai biến chứa thông tin như biểu diễn một ảnh. Các mô hình biểu diễn ảnh cho thấy một mô tả logic hay định lượng các tính chất của hàm.

Trong biểu diễn ảnh cần chú ý tới đặc tính trung thực của ảnh hoặc các tiêu chuẩn thông minh để đo chất lượng của ảnh hoặc tính hiệu quả của các kỹ thuật xử lý.

Một số mô hình thường được dùng để biểu diễn ảnh như: mô hình toán học, mô hình thống kê. Trong mô hình toán học, ảnh hai chiều biểu diễn nhờ các hàm hai biến trực giao gọi là hàm cơ sở. Với mô hình thống kê, một ảnh được coi như một phần tử của một tập hợp đặc trưng bởi các đại lượng như: kỳ vọng toán học, hiệp biến, phương sai và mômen.

1.4.8. Nâng cao chất lượng ảnh

Tăng cường ảnh, khôi phục ảnh (hay nâng cao chất lượng ảnh) là một bước quan trọng tạo tiền đề cho xử lý ảnh. Mục đích là làm nổi bật một số đặc tính của ảnh: Thay đổi độ tương phản, lọc nhiễu, nổi biên, làm trơn biên, khuếch đại ảnh.

- Tăng cường ảnh: Nhằm hoàn thiện trạng thái quan sát của một ảnh. Bao gồm điều khiển mức xám, thay đổi độ tương phản, giảm nhiễu, làm trơn, nội suy.
- Khôi phục ảnh: Nhằm khôi phục ảnh gần với trạng thái thực nhất trước khi biến dạng, tùy theo nguyên nhân gây ra biến dạng.

Phạm vi đồ án sẽ tập trung vào việc khôi phục một bức ảnh bị mờ do chuyển động hoặc rung lắc. Đây cũng chính là tiền đề để tiền xử lý hình ảnh phục vụ cho các tác vụ quan trọng khác trong nhiều lĩnh vực sau này.

Hình 1.11. Cải thiện chất lượng ảnh bằng khử nhiễu

1.4.9. Biến đổi ảnh

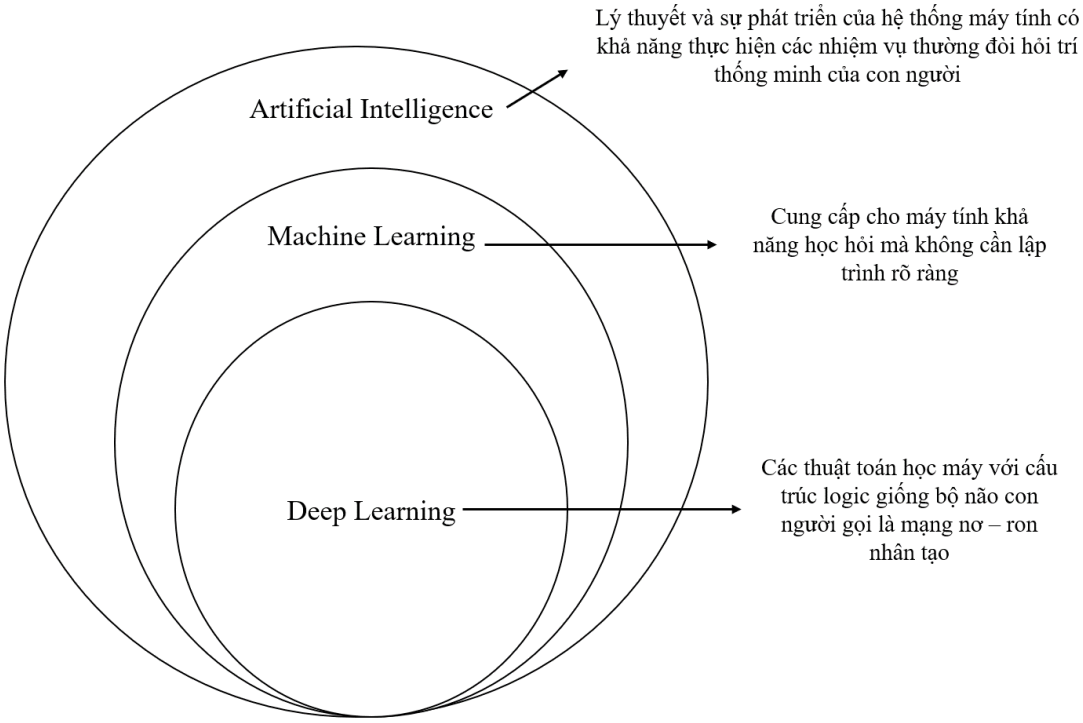
Trong việc xử lý ảnh số, biến đổi ảnh là một trong những thao tác quan trọng giúp thay đổi trạng thái ảnh, chuyển đổi ảnh sang một hệ quy chiếu khác hay trích lọc ra những đặc trưng quan trọng của ảnh. Hiện nay, việc biến đổi ảnh được thông qua rất nhiều thuật toán, bộ lọc nhằm phục vụ mục đích cho nhiều loại bài toán cụ thể. Các biến đổi thường gặp trong xử lý ảnh bao gồm:

- Biến đổi Fourier (Fourier Transform)
- Biến đổi KL (Karhunen-Loeve Transform)
- Biến đổi Wavelet (Wavelet Transform)
- Các phép biến đổi dựa trên toán tử tuyến tính
- Phép tích chập (2D Convolution)

1.5. Giới thiệu về Deep Learning

1.5.1. Khái niệm cơ bản về Deep Learning

Deep Learning - học sâu có thể được xem là một lĩnh vực con của Machine Learning - học máy, ở đó các máy tính sẽ học và cải thiện chính nó thông qua các thuật toán. Deep Learning được xây dựng dựa trên các khái niệm phức tạp hơn rất nhiều, chủ yếu hoạt động với các mạng nơ-ron nhân tạo để bắt chước khả năng tư duy và suy nghĩ của bộ não con người. [4]



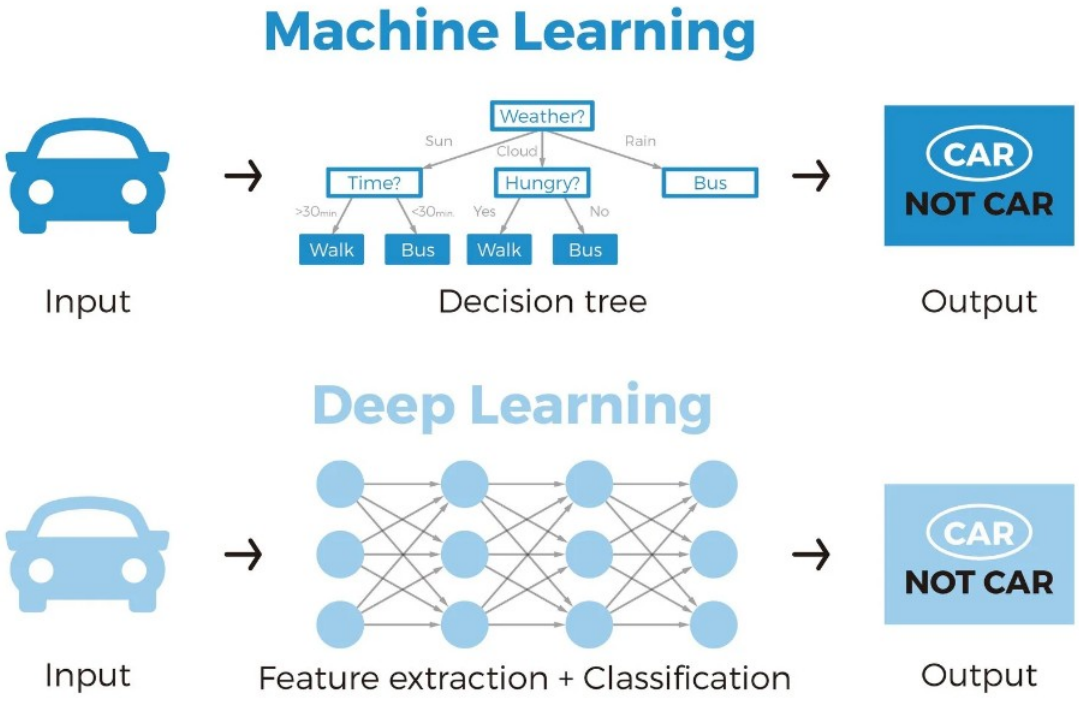
Hình 1.12. Ảnh minh họa Deep Learning

1.5.2. Deep Learning hoạt động như thế nào

Deep Learning là một phương pháp được xây dựng dựa trên nền tảng của Machine Learning. Các mạng nơ-ron nhân tạo trong Deep Learning được xây dựng dưa trên ý tưởng là bộ não con người với các nơ – ron thần kinh liên kết chặt chẽ với nhau.

Các nơ – ron trong một mạng học sâu Deep Learning liên kết giữa các node từ lớp này sang lớp khác. Trong một mạng nơ – ron, số lượng các lớp càng nhiều thì mạng sẽ càng sâu. Mỗi liên kết giữa các node sẽ tương ứng với một trọng số, trọng số này càng cao thì càng ảnh hưởng đến kết nối của mạng nơ – ron đó.

Để thực hiện, các nơ – ron sẽ có hàm kích hoạt tương ứng và nhiệm vụ của các hàm kích hoạt này sẽ giúp mạng lưới lấy ra những đặc trưng của dữ liệu để từ đó mô hình có thể tiếp thu, học hỏi được từ dữ liệu đó. Dữ liệu được người dung đưa vào mạng nơ – ron sẽ đi qua các layer và trả về layer cuối cùng gọi là output layer.



Hình 1.13. Ảnh minh họa hoạt động giữa Machine Learning và Deep Learning

Trong quá trình huấn luyện mô hình mạng nơ-ron, các trọng số sẽ được thay đổi và nhiệm vụ của mô hình là tìm ra bộ giá trị của trọng số sao cho phán đoán là tốt nhất.

Các hệ thống Deep Learning yêu cầu phần cứng phải rất mạnh để có thể xử lý được lượng dữ liệu lớn và thực hiện các phép tính phức tạp. Nhiều mô hình Deep Learning có thể mất nhiều tuần, thậm chí nhiều tháng để triển khai trên những phần cứng tiên tiến nhất hiện nay.

1.5.3. Ưu – nhược điểm của Deep Learning

Ưu điểm: Deep Learning là một bước ngoặt to lớn trong lĩnh vực trí tuệ nhân tạo, cho phép khác nhà khoa học dữ liệu xây dựng nhiều mô hình có độ chính xác rất cao trong lĩnh vực nhận dạng ảnh, xử lý ngôn ngữ tự nhiên, xử lý giọng nói. Một số ưu điểm vượt trội của Deep Learning gồm có:

- Kiến trúc mạng nơ-ron linh hoạt, có thể dễ dàng thay đổi để phù hợp với nhiều vấn đề khác nhau.
- Có khả năng giải quyết nhiều bài toán phức tạp với độ chính xác rất cao.
- Tính tự động hoá cao, có khả năng tự điều chỉnh và tự tối ưu.
- Có khả năng thực hiện tính toán song song, hiệu năng tốt, xử lý được lượng dữ liệu lớn.

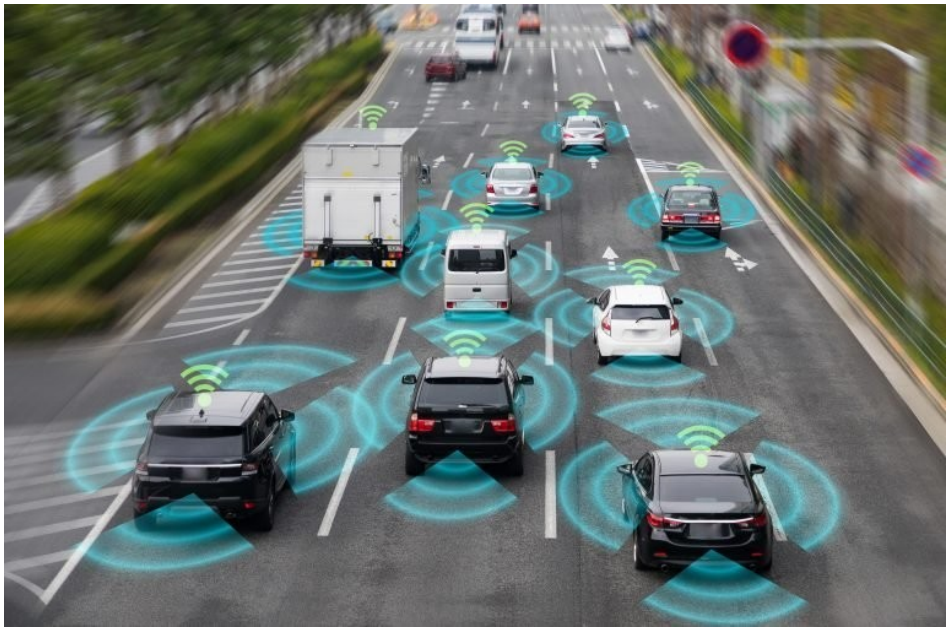
Nhược điểm: Bên cạnh những ưu điểm, mặt khác, hiện nay Deep Learning vẫn còn nhiều khó khăn và hạn chế, chẳng hạn như:

- Cần có khối lượng dữ liệu rất lớn để tận dụng tối đa khả năng của Deep Learning.
- Chi phí tính toán cao vì phải xử lý nhiều mô hình phức tạp.
- Chưa có nền tảng lý thuyết mạnh mẽ để lựa chọn các công cụ tối ưu cho Deep Learning.

1.5.4. Ứng dụng của Deep Learning

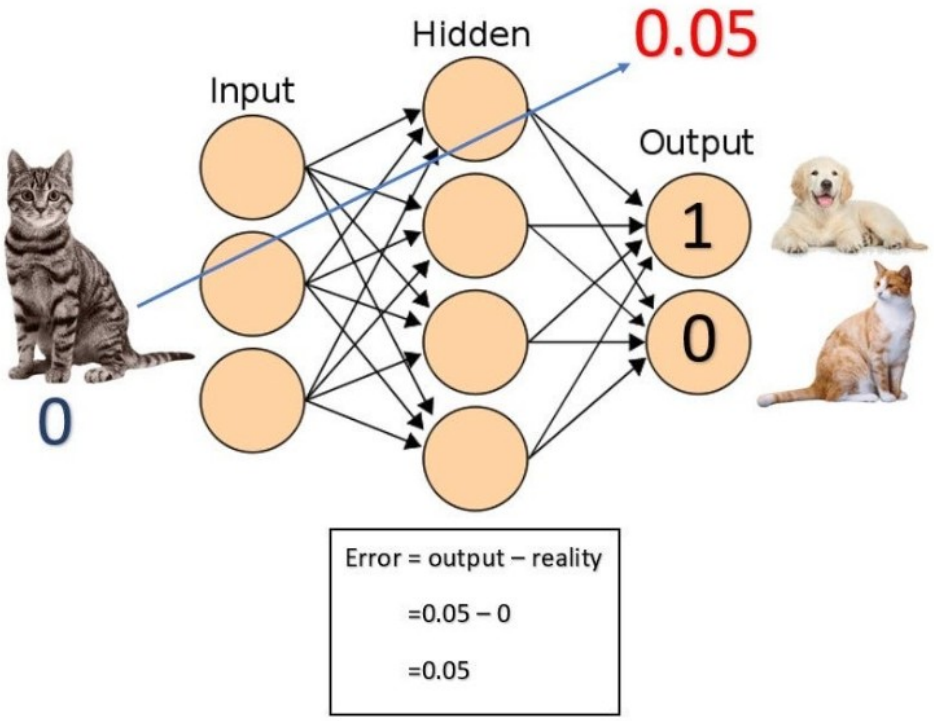
Kiến trúc mạng nơ-ron trong Deep Learning được ứng dụng trong các công việc yêu cầu sức mạnh tính toán cao, xử lý nhiều dữ liệu và độ phức tạp lớn. Dưới đây là một số ứng dụng của Deep Learning trong cuộc sống ngày nay mà ta hay được bắt gặp nhất:

- **Xử lý ảnh**: Với sự phát triển vượt bậc của Deep Learning ngày nay thì các kỹ thuật Deep Learning có thể hoàn toàn được áp dụng vào để thực hiện các tác vụ liên quan đến xử lý hình ảnh. Có rất nhiều ứng dụng của Deep Learning trong lĩnh vực xử lý ảnh rộng lớn này có thể kể đến các bài toán phát hiện cạnh, phát hiện vật thể, đánh dấu vật thể hay là bài toán xử lý ảnh mờ của em cũng ứng dụng kỹ thuật Deep Learning để thực hiện. Việc xử lý ảnh áp dụng Deep Learning đã dần dần trở thành một lĩnh vực rộng lớn trong thế giới khoa học máy tính đem lại rất nhiều hiệu quả cho con người trong lao động, sản xuất, giáo dục.
- **Xe tự lái:** Một trong những công nghệ mới và hấp dẫn nhất hiện nay là xe tự động lái, nó được xây dựng dựa trên các mạng nơ-ron cấp cao. Nói một cách đơn giản, các mô hình Deep Learning sẽ nhận diện các đối tượng ở môi trường xung quanh xe, tính toán khoảng cách giữa xe và các phương tiện khác, xác định vị trí làn đường, tín hiệu giao thông,… từ đó đưa ra được các quyết định tối ưu và nhanh chóng nhất. Một trong những hãng xe tiên phong trong việc sản xuất xe tự lái hiện nay là Tesla.



Hình 1.14. Hình ảnh minh họa xe tự lái

- **Phân loại sắc thái:** Đây là lĩnh vực phân tích cảm xúc của con người thông qua việc xử lý ngôn ngữ tự nhiên, phân tích văn bản và thống kê. Các công ty có thể ứng dụng Deep Learning để hiểu và phán đoán cảm xúc của khách hàng dựa trên những đánh giá, bình luận, tweet,… từ đó đưa ra những chiến lược kinh doanh và marketing phù hợp với từng nhóm đối tượng. Ngoài ra, phân loại sắc thái còn xuất hiện rất nhiều trong cái bài toán xử lý ảnh, phân biệt nội dung thể hiện chính trong những bức ảnh giúp chương trình nhận biết hoặc xử lý các tác vụ cụ thể khác.



Hình 1.15. Hình ảnh minh họa bài toán phân loại

- **Trợ lý ảo:** Trợ lý ảo đang được ứng dụng rất nhiều trong đời sống hàng ngày, trong đó phổ biến gồm có chatbot, giảng viên online, Google Assistant, Siri, Cortana. Các trợ lý ảo được xây dựng dựa trên Deep Learning với các thuật toán nhận diện văn bản, xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói,…

Hình 1.16. Hình ảnh về trợ lý ảo Siri của Apple

- **Mạng xã hội:** Một số nền tảng mạng xã hội lớn như Twitter cũng ứng dụng các thuật toán Deep Learning để cải thiện các dịch vụ của mình. Cụ thể, những trang này sẽ phân tích một lượng lớn dữ liệu thông qua mạng nơ-ron nhân tạo để tìm hiểu về các tuỳ chọn của người dùng. Ngoài ra, Instagram cũng sử dụng Deep Learning để tránh các hành vi bạo lực trên không gian mạng, chặn các bình luận vi phạm, không phù hợp. Facebook cũng không nằm ngoài danh sách các mạng xã hội ứng dụng Deep Learning vào sản phẩm của mình. Các thuật toán mạng nơ-ron sâu được sử dụng để gợi ý trang, bạn bè, dịch vụ, nhân diện khuôn mặt.

Hình 1.17. Hình ảnh minh họa các nền tảng mạng xã hội

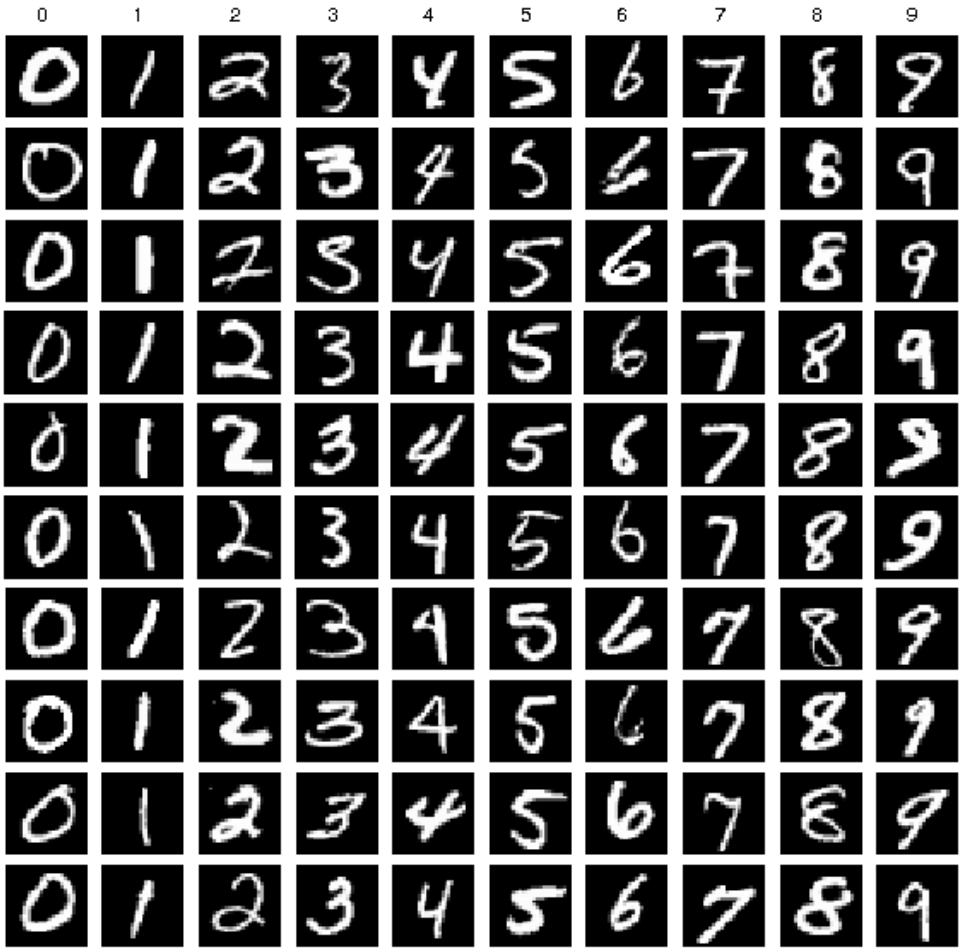
- **Chăm sóc sức khỏe:** Deep Learning cũng có đóng góp không nhỏ vào lĩnh vực y tế, trong đó phổ biến gồm có các mô hình dự đoán tình trạng bệnh, chẩn đoán ung thư, phân tích kết quả MRI, X-ray.

Hình 1.18. Hình ảnh minh họa robot thông minh trong y tế

1.5.5. Dữ liệu trong Deep Learning

Bùng nổ dữ liệu lớn trong những năm gần đây đã giúp việc xây dựng các mô hình Deep Learning trở nên dễ dàng hơn. Tuy nhiên, đây vẫn là một lĩnh vực vô cùng phức tạp và tốn kém. Vì phải xử lý lượng dữ liệu vô cùng lớn nên các mô hình Deep Learning thường rất nặng về mặt tài nguyên tính toán và GPU để có được hiệu năng tốt nhất.

Các thuật toán Deep Learning có thể tìm ra được các mối quan hệ ẩn sâu trong những bộ dữ liệu. Tuy nhiên việc này cũng đồng nghĩa với lượng dữ liệu đầu vào (dữ liệu đã được gán nhãn) phải đủ lớn. Việc gán nhãn dữ liệu cũng đòi hỏi nhiều nguồn lực và thời gian , đặc biệt là trong lĩnh vực y tế phải yêu cầu chuyên môn cao mới có khả năng gán nhãn dữ liệu chính xác. Chính vì vậy, công đoạn chuẩn bị dữ liệu cho các mô hình Deep Learning học tập là công đoạn đòi hỏi rất nhiều chi phí và thời gian. Bên cạnh đó, các dữ liệu thu thập được cần phải xử lý, chắc lọc mới có thể cho vào để mô hình học. Công việc này đòi hỏi rất nhiều thời gian, sự kiên nhẫn và nhiều lúc cần tới cả chuyên môn của những người hành nghề trong lĩnh vực mà mô hình được ứng dụng.

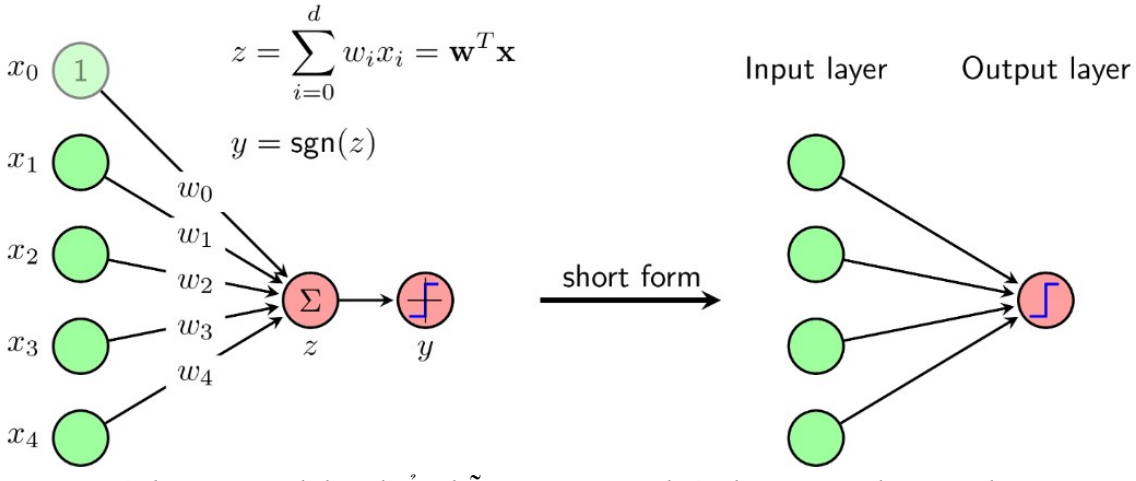


Hình 1.19. Hình ảnh minh họa tập dữ liệu chữ số viết tay MNIST dataset [5]

1.6. Mạng nơ – ron

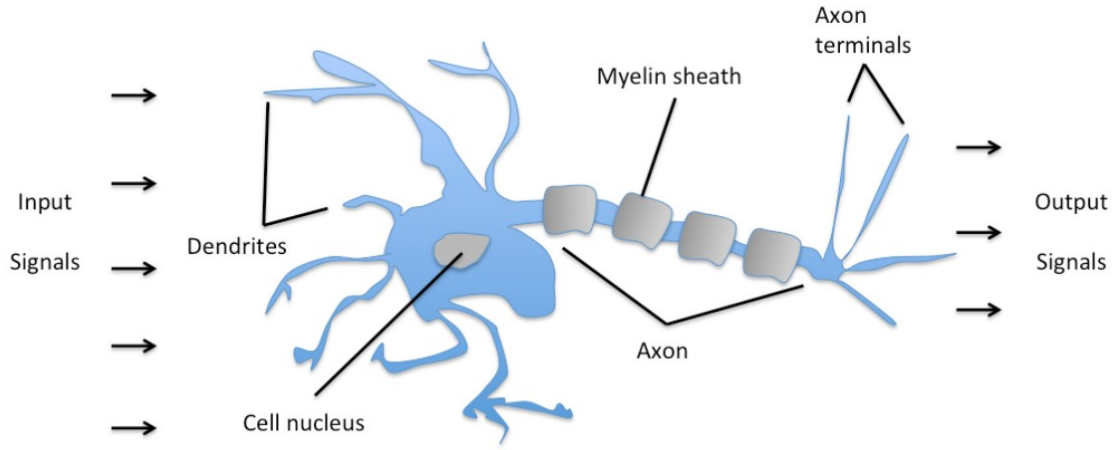
1.6.1. Mạng nơ – ron nhân tạo

Mạng nơ-ron nhân tạo (Neural Network - NN) là một mô hình lập trình mô phỏng cách thức hoạt động của mạng nơ-ron thần kinh. Nó được cấu thành bởi các nơ-ron đơn lẻ được gọi là các perceptron.



Hình 1.20. Minh họa biểu diễn Perceptron dưới dạng Neural Network [6]

❖ *Perceptron*

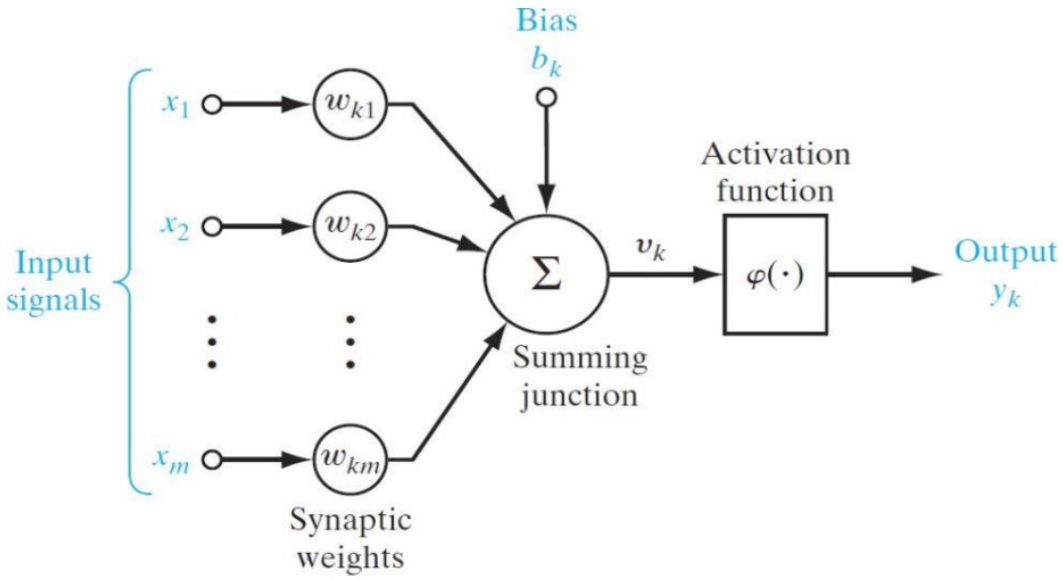


Hình 1.21. Hình ảnh minh họa nơ-ron thần kinh sinh học [6]

Một nơ-ron sinh học có phần thân (cell body) chứa nhân (nucleus), các tín hiệu đầu vào qua sợ nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác.

Mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua các sợi trục, đến các sợi nhánh của các nơ-ron khác. Mỗi nơ-ron nhận các xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác.

Mô hình của perceptron cũng vậy:



Hình 1.22. Hình ảnh minh họa kiến trúc nơ -ron nhân tạo cơ bản nhất [7]

Một perception sẽ nhận một hoặc nhiều đầu vào x được thể hiện bằng các node và cho ra một kết quả dạng nhị phân. Các đầu vào được điều phối tầm ảnh

hưởng bởi các tham số trọng lượng tương ứng w của nó, còn kết quả đầu ra được quyết định dựa vào một ngưỡng b quyết định nào đó:

$$y=\begin{cases}0, if \sum_i w_i . x_i \leq threshold \\ 1, if \sum_i w_i . x_i > threshold\end{cases}$$

Nếu ta đặt b = -threshold (ngưỡng) thì công thức trên có thể viết lại thành:

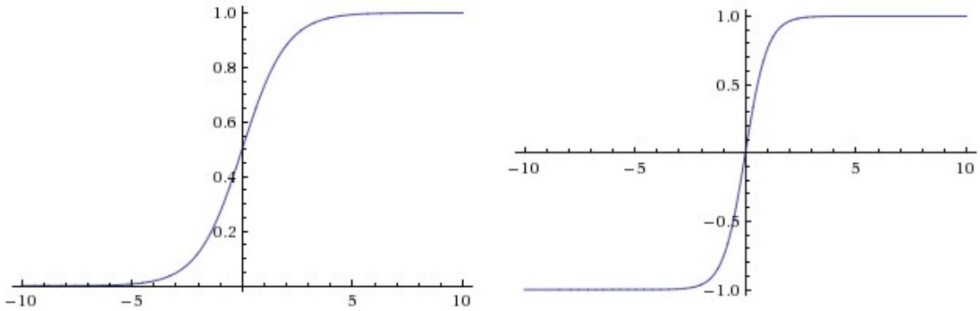
$$y=\begin{cases}0, if \sum_i w_i . x_i + b \leq 0 \\ 1, if \sum_i w_i . x_i + b > 0\end{cases}$$

Tiếp đó, nếu ta đặt x_0=1 và w_0=b, thì công thức trên có thể viết lại thành:

$$y=\begin{cases}0, if\ w^T x \leq 0 \\ 1, if\ w^T x > 0\end{cases}$$

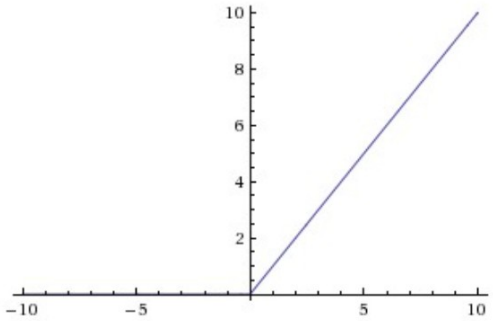
Với đầu ra là dạng nhị phân, rất khó có thể điều chỉnh một lượng nhỏ đầu vào để đầu ra thay đổi chút ít, nên để linh động mở rộng đầu ra thành khoảng [0, 1]. Lúc này đầu ra được quyết định bởi một hàm $\sigma(w^T x)$ có giá trị trong khoảng [0, 1] gọi là hàm kích hoạt (activation function). Các hàm kích hoạt có thể kể đến như:

- *Sigmoid và tanh*



Hình 1.23. Minh họa đồ thị của hàm kích hoạt Sigmoid và Tanh [8]

- *ReLU*

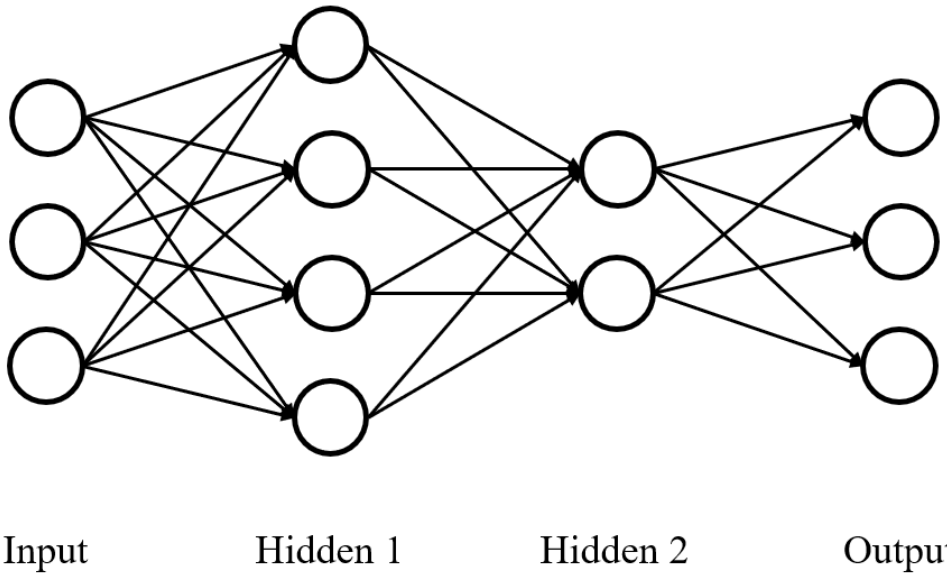


Hình 1.24. Hình ảnh minh họa đồ thị hàm kích hoạt ReLU [9]

❖ *Multi – layer perceptron*

Khi kết hợp các tầng perceptron, có một mạng nơ-ron tổng quát gồm 3 layer:

- Input layer: nhiều node ở input tạo thành một lớp đầu vào của mạng.
- Hidden layer: các layer trung gian thể hiện cho việc suy luận logic của mạng.
- Output layer: nhiều node ở đầu ra tạo thành một lớp đầu ra.



Hình 1.25.Multi – layer Perceptron với 2 hidden layer

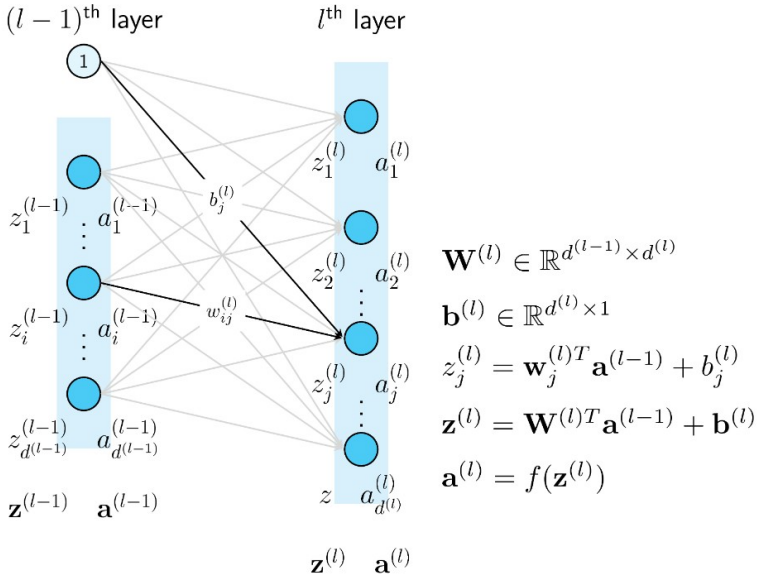
Số lượng layer trong một MLP được tính bằng số hidden layers cộng với 1. Tức là khi đếm số layers của một MLP, ta không tính input layers. Số lượng layer trong một MLP thường được ký hiệu là L. Đối với hình 24 thì $L=3$.

Mỗi node trong hidden layer và output layer:

- Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
- Mỗi node có 1 hệ số bias b riêng.
- Diễn ra 2 bước: tính tổng linear và áp dụng hàm kích hoạt

❖ *Units*

Ở những hình ảnh minh họa bên trên có xuất hiện những node tròn trong các layer, mỗi node hình tròn này được gọi là một unit. Các unit ở các layer như input layer, hidden layer, output layer được gọi lần lượt là các input unit, hidden unit và output unit. Đầu vào của các hidden layer được ký hiệu bởi z, đầu ra của mỗi unit thường được ký hiệu là a (thể hiện là hàm kích hoạt – activation function). Điều này có nghĩa là mỗi unit khi nhận đầu vào là kết quả z của hàm tuyến tính sẽ được áp dụng các hàm kích hoạt lên z và cho ra kết quả là giá trị a. Đầu ra của unit thứ i trong layer thứ l được ký hiệu là $a_i^{(l)}$.



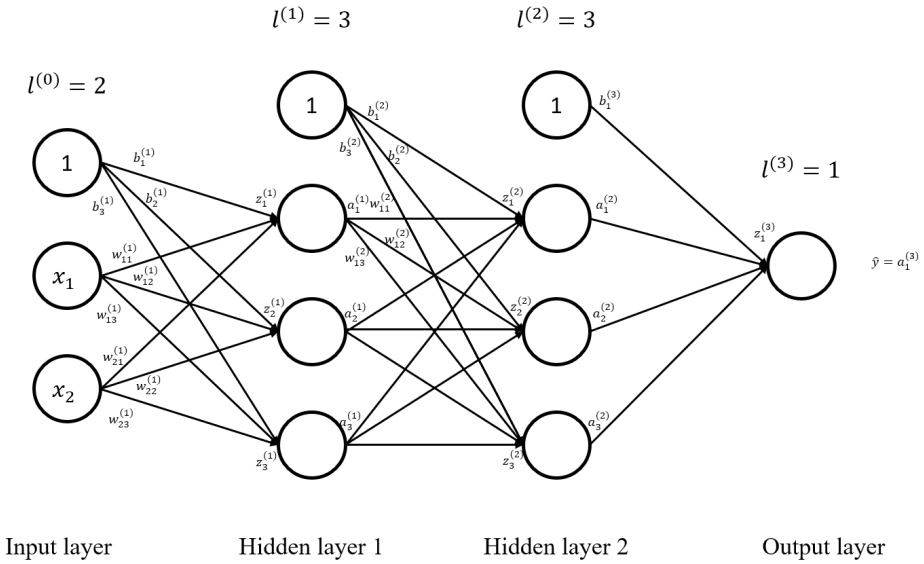
Hình 1.26. Hình ảnh minh họa các ký hiệu sử dụng trong mạng nơ -ron [8]

❖ *Weights và Biases*

Giữa các layer với nhau sẽ có một ma trận trọng số, hay nói cách khác nếu ta không tính input layer thì một MLP có L layer thì sẽ có L ma trận trọng số. Các ma trận này được ký hiệu là $W^{(l)} \in R^{d^{l-1} \times d^l}, l=1,2,\ldots,L$ trong đó $W^{(l)}$ thể hiện các kết nối từ layer thứ $l-1$ tới layer thứ l. Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của layer thứ $(l-1)$ tới node thứ j của layer thứ (l). Các bias của layer thứ (l) được ký hiệu là $b^{(l)} \in R^{d^{(l)}}$. Các trọng số này được ký hiệu như trên hình 25. Mục tiêu của các mô hình Deep Learning là đi tối ưu một MLP cho tác vụ cụ thể nào đó và việc cần phải làm đó là đi tìm các weights và biases này.

Tập hợp các weights và biases lần lượt được ký hiệu là W và b.

❖ *Feedforward*



Hình 1.27. Ví dụ minh họa về mạng nơ – ron.

Quá trình feedforward sẽ được em dựa vào hình ảnh trên để mô tả như sau:

Gọi input layer ở hình trên là $a^{(0)} = x$ (ở hình 1.27 thì $a^{(0)} = \{x_1; x_2\}$), thực hiện tính $z^{(l)}$ là đầu ra của input layer. Đồng thời đưa $z^{(l)}$ qua hàm kích hoạt tại các node ở hidden layer để được $a^{(l)}$:

$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} a_1^{(0)} * w_{11}^{(1)} + a_2^{(0)} * w_{21}^{(1)} + b_1^{(1)} \\ a_1^{(0)} * w_{12}^{(1)} + a_2^{(0)} * w_{22}^{(1)} + b_2^{(1)} \\ a_1^{(0)} * w_{13}^{(1)} + a_2^{(0)} * w_{23}^{(1)} + b_3^{(1)} \end{bmatrix}$$

$$= \left(W^{(1)}\right)^T * a^{(0)} + b^{(1)}$$

$$=> \quad a^{(1)} = \sigma(z^{(1)})$$

Tương tự sau đó, ta có các kết quả tính của mỗi lớp:

$$z^{(2)} = \left(W^{(2)}\right)^T * a^{(1)} + b^{(2)}$$

$$a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = \left(W^{(3)}\right)^T * a^{(2)} + b^{(3)}$$

$$\hat{y} = \; a^{(3)} = \sigma(z^{(3)})$$

Quá trình tính toán trên có thể được mô tả tổng quát như hình dưới đây:

$$a^{(0)} \longrightarrow z^{(1)} \longrightarrow a^{(1)} \longrightarrow z^{(2)} \longrightarrow a^{(2)} \longrightarrow z^{(3)} \longrightarrow a^{(3)} = \hat{y}$$

Hình 1.28. Mô tả thứ tự tính toán của quá trình feedfoward

Có thể thấy rằng, quá trình tính toán được thực hiện lần lượt từ trái qua phải và đưa ra kết quả dự đoán cuối cùng. Quá trình này được gọi là feedforward,

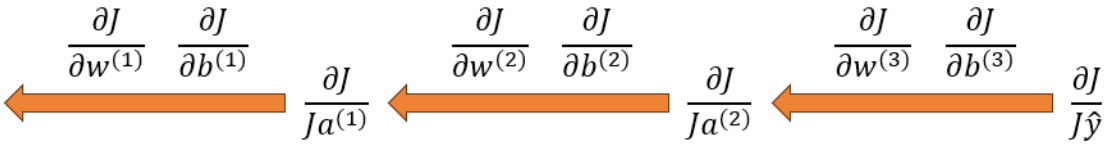
đưa đầu vào dữ liệu vào nhằm dự đoán thử các kết quả để có hướng điều chỉnh các tham số weights và biases sao cho phù hợp nhất.

❖ *Backpropagation*

Từ quá trình feedforward để đưa ra kết quả dự đoán thì làm thế nào để ta có thể điều chỉnh các tham số weights và biases cho mô hình trở lên phù hợp với bài toán. Dưới đây, em sẽ trình bày về quá trình backpropagation, là cách thức để mô hình tự điều chỉnh các tham số weights và bias sao cho phù hợp.

Từ input X ta có thể tính được giá trị dự đoán $y^{'}$, sau đó tùy theo từng bài toán cụ thể mà tính toán loss function (hàm mất mát thể hiện sự chênh lệch giữa kết quả thực và kết quả dự đoán. Từ kết quả của hàm loss function ta sẽ thực hiện các phép tính toán tối ưu để điều chỉnh các tham số weights và bias. Cụ thể ở đây, ta sẽ áp dụng kiến thức về Gradient Descent để tối ưu cho hàm mất mát (loss function) về giá trị nhỏ nhất – có nghĩa là sự chênh lệch giữa kết quả dự đoán và kết quả thực tế là thấp nhất.

Ví dụ đơn giản về hàm loss function như sau: $J=\frac{1}{N}\times(\sum_{i=1}^{N}(y_i^{'}-y_i)^2)$, trong đó J thể hiện cho kết quả chênh lệch của hàm mất mát (loss function), N là số điểm dữ liệu đầu vào, $y_i^{'}$ là kết quả dự đoán của dữ liệu thứ i và y_i là kết quả thực của dữ liệu thứ i.



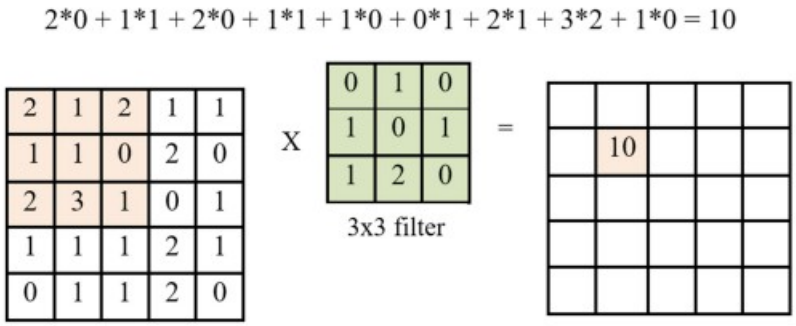
Hình 1.29. Minh họa cho quá trình backpropagation

Như vậy, mạng nơ-ron nhân tạo là một mô hình được tạo nên từ một lượng lớn các phần tử (nơ-ron) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể như phân loại dữ liệu, nhận dạng mẫu, thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học ở đây chính là quá trình hiệu chỉnh trọng số liên kết giữa các nơ-ron.

1.6.2. Kiến trúc mạng nơ – ron tích chập

❖ *Tích chập*

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



Hình 1.30. Minh họa cho phép tích chập trong không gian 2 chiều

Ma trận bên trái là một ảnh xám, mỗi giá trị của ma trận tương đương với một điểm ảnh (pixel) có giá trị biến thiên từ 0 đến 255. Sliding window còn có tên gọi là kernel, filter hay feature detector. Ở đây, ta dùng một ma trận filter kích thước 3×3 nhân từng thành phần tương ứng với ma trận ảnh bên trái. Giá trị đầu ra do tích của các thành phần này cộng lại. Kết quả của tích chập là một ma trận sinh ra từ việc trượt ma trận filter và thực hiện tích chập cùng lúc lên toàn bộ ma trận ảnh bên trái.

Nguyên tắc hoạt động của bộ lọc (filter) nó sẽ trượt dần từ trái qua phải từ trên xuống dưới cho đến khi toàn bộ các điểm ảnh được duyệt qua. Khoảng cách mà bộ lọc di chuyển từ điểm ảnh này sang điểm ảnh khác được gọi là Stride (bước nhảy). Khi Stride = 1 thì các bộ lọc sẽ di chuyển mỗi lần dịch sang phải 1 pixel. Tương tự nếu Stride = 2 thì bộ lọc sẽ di chuyển mỗi lần dịch sang phải 2 pixel. Việc lựa chọn Stride càng lớn thì chiều của kết quả tích chập sẽ càng giảm. Ví dụ đối với hình 29, nếu ta chọn Stride = 1 thì kết quả đầu ra sau khi tích chập sẽ là một ma trận có kích thước 3×3 và nếu ta lựa chọn Stride = 2 thì sẽ được kết quả đầu ra là ma trận kích thước 2×2.

Trên đây là giới thiệu cách tính tích chập, tiếp theo đồ án sẽ đi tới giới thiệu các thành phần cơ bản của một mạng nơ – ron tích chập.

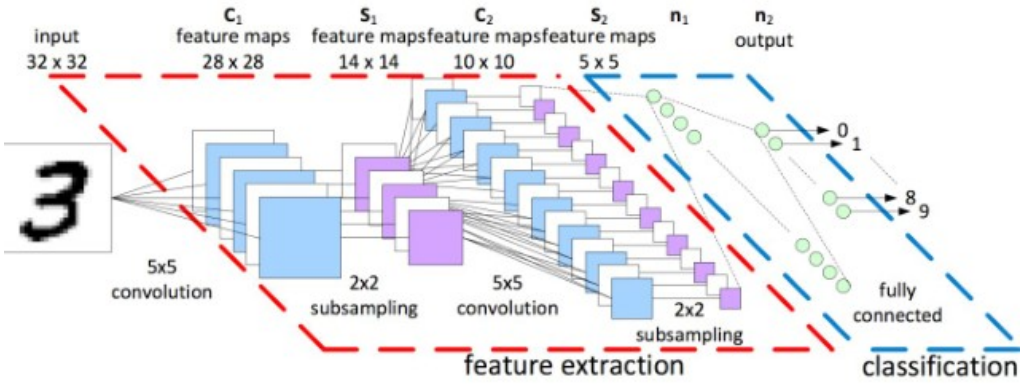
❖ *Giới thiệu mạng nơ – ron tích chập*

Trong các bài toán xử lý ảnh với Deep Learning là việc một tấm ảnh đầu vào mang dữ liệu quá lớn khiến cho các mô hình Deep Learning có xu hướng kém hiệu quả. Ví dụ với một tấm ảnh màu kích thước $32 \times 32 \times 3$ (cao 32 pixel, rộng 32 pixel và có 3 kênh màu R-G-B) thì đầu vào của một mạng nơ – ron truyền thống với input layer đã có 3072 trọng số. Con số 3072 vẫn hoàn toàn có thể kiểm soát được và một máy tính bình thường vẫn có thể xử lý nhưng khi bức ảnh trở lên lớn hơn thì lại là chuyện khác. Chả hạn như một tấm ảnh màu với kích thước $200 \times 200 \times 3$, đã khiến cho mô hình có 120.000 trọng số ở layer đầu tiên, cộng thêm các trọng số ở các hidden layer sau sẽ khiến số lượng trọng số của mạng trở lên rất lớn và máy tính không thể thực hiện các tính toán hoặc có thể nhưng tốn rất nhiều thời gian. Chính điều này, đã khiến việc xử lý ảnh với

các bức ảnh có độ phân giải cao trở nên khó khăn và tiêu tốn rất nhiều tài nguyên tính toán.

Dựa trên những khó khăn trong việc xử lý ảnh đó, Convolution Neural Network (mạng nơ-ron tích chập) ra đời để giải quyết vấn đề này. Bắt đầu từ ý tưởng trong một ảnh thì các pixel ở cạnh nhau sẽ có liên kết, ảnh hưởng tới nhau nhiều hơn các pixel ở xa cho nên CNN sẽ tìm cách thu gọn các pixel gần nhau thành 1 pixel tùy theo mục đích của bài toán. Việc làm này giúp giảm đi đáng kể số lượng trọng số ban đầu của một bức ảnh mà vẫn giữ lại được các đặc trưng của ảnh.

Một mạng nơ – ron tích chập bao gồm tập hợp các lớp chính cơ bản gồm: lớp tích chập (Convolution layer), lớp lấy mẫu (Pooling layer) và lớp kết nối đầy đủ (Fully connected layer).



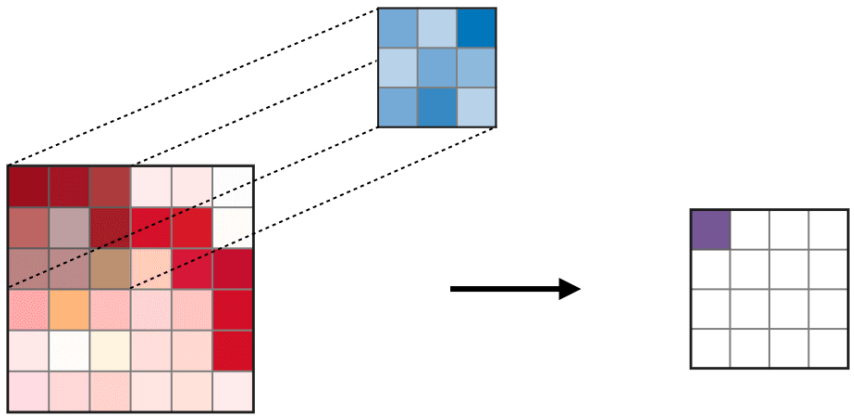
Hình 1.31. Minh họa tích chập cho bài toán phân loại

Trong mô hình CNN, các layer liên kết được với nhau thông qua cơ chế convolution. Layer tiếp theo là kết quả convolution từ layer trước đó. Nhờ vậy mà ta có được các kết nối cục bộ. Nghĩa là mỗi nơ-ron ở layer tiếp theo sinh ra từ filter áp đặt lên một vùng ảnh cục bộ của nơ-ron layer trước đó. Mỗi layer như vậy được áp đặt các filter khác nhau, thông thường có vài trăm đến vài nghìn filter như vậy. Một số layer khác như pooling/subsampling layer dùng để chắt lọc lại các thông tin hữu ích hơn (loại bỏ các thông tin nhiễu).

CNN có tính bất biến và tính kết hợp cục bộ (Location Invariance and Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các góc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể. Pooling layer sẽ cho tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling). Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter. Đó là lý do tại sao CNN cho ra mô hình với độ chính xác rất cao. Tiếp theo, em sẽ trình bày chi tiết các lớp trong mô hình CNN.

- ❖ *Lớp tích chập*

Layer này chính là nơi thể hiện tư tưởng ban đầu của CNN. Thay vì kết nối toàn bộ điểm ảnh, layer này sẽ sử dụng một tập các bộ lọc (filters) có kích thước nhỏ so với ảnh (thường là 5×5 hoặc 3×3) áp vào một vùng trong ảnh và tiến hành tính tích chập giữa bộ lọc và giá trị điểm ảnh trong vùng cục bộ đó. Bộ lọc sẽ lần lượt được dịch chuyển theo một giá trị bước trượt (stride) chạy dọc theo ảnh và quét toàn bộ ảnh.



Hình 1.32. Minh họa thực hiện tích chập với bộ lọc [10]

Trong hình 1.32, bộ lọc được sử dụng là một ma trận có kích thước 3×3, bộ lọc này sẽ dịch chuyển lần lượt từ trái qua phải, từ trên xuống dưới qua từng vùng ảnh đến khi quét hết toàn bộ bức ảnh.

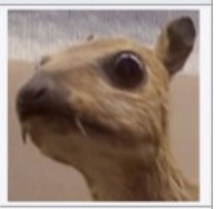
Kích thước tấm ảnh đầu ra sẽ được xác định tùy theo kích thước của bước nhảy (stride), các khoảng trắng được thêm (padding) và kích thước tấm lọc filter theo công thức:

$$O = \frac{i + 2 * p - k}{s} + 1$$

Trong đó:

- i là kích thước ảnh đầu vào (thường là ảnh vuông)
- p là kích thước khoảng trắng viền được thêm vào
- k là kích thước bộ lọc filter
- s là bước nhảy của bộ lọc

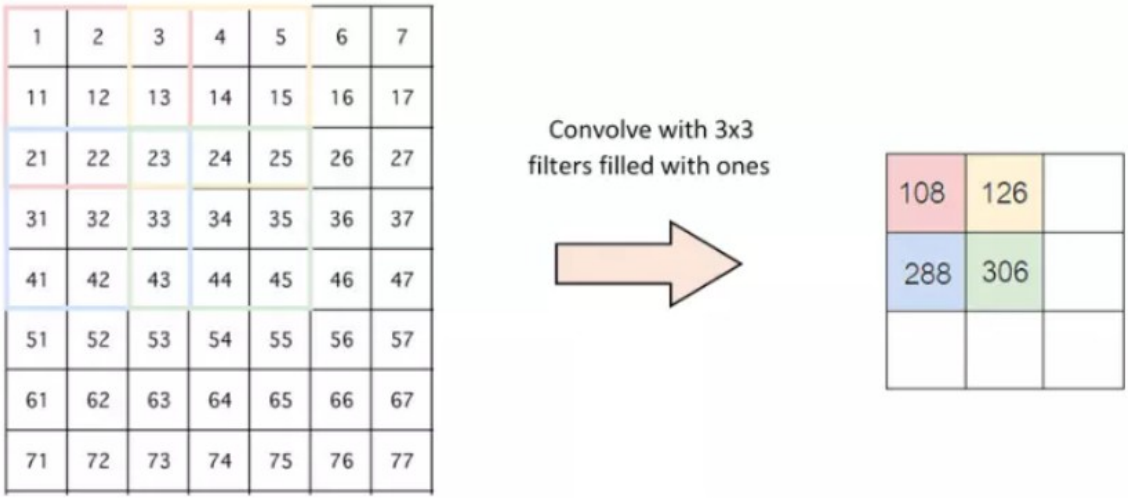
Khi đưa ảnh vào lớp tích chập, đầu ra của nó ứng với một loạt điểm ảnh. Có thể thực hiện các hoạt động như phát hiện cạnh, làm mờ và làm sắc nét bằng các kết hợp hình ảnh đầu vào với các bộ lọc khác nhau. Các trọng số của các bộ lọc này được khởi tạo ngẫu nhiên và cập nhật trong quá trình huấn luyện.

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Hình 1.33. Minh họa kết quả tích chập với các bộ lọc khác nhau [3]

❖ *Bước nhảy (Stride)*

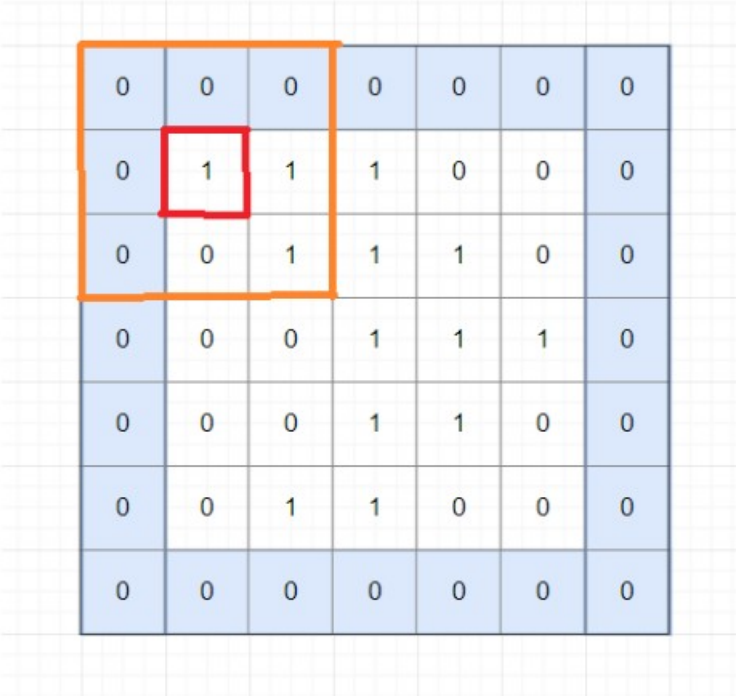
Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các filter 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride là 2.



Hình 1.34. Hình ảnh minh họa stride = 2

❖ *Đường viền*

Quá trình tích chập (convolution) có thể làm giảm bớt đi kích thước của ảnh ban đầu. Điều này có thể dẫn tới việc tích chập có thể bỏ qua những thông tin quan trọng của một tấm ảnh làm cho kết quả tính toán trở nên sai sót. Hoặc đôi khi bộ lọc filter không phù hợp với hình ảnh đầu vào và chúng ta cần giải quyết các vấn đề này. Chính vì thế, việc chèn thêm các số 0 và 4 đường biên của hình ảnh sẽ giúp ta cải thiện được các vấn đề trên. Việc làm này vừa giúp tăng kích thước bức ảnh để đảm bảo đầu ra tích chập là một tấm ảnh bằng với tấm ảnh ban đầu, vừa có thể giúp tấm ảnh phù hợp với các bộ lọc khác.

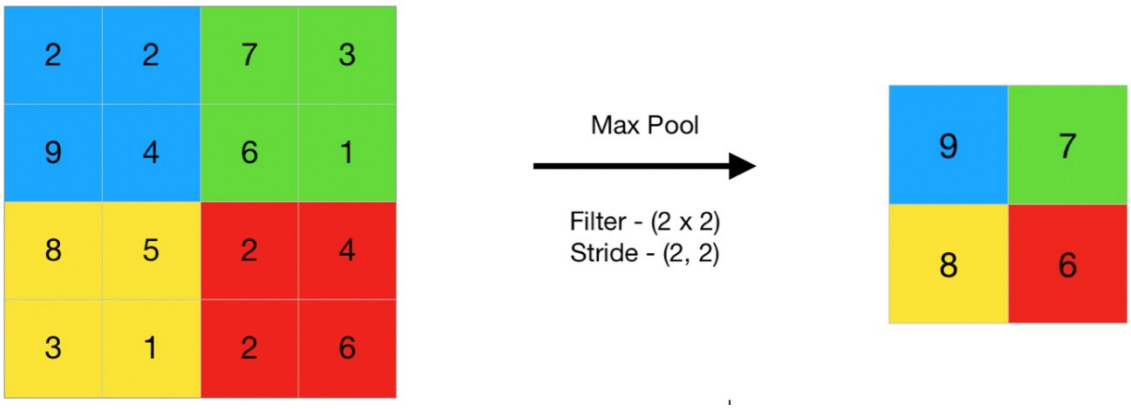


Hình 1.35. Hình ảnh minh họa ma trận ảnh được padding = 1

❖ *Lớp lấy mẫu (Pooling)*

Layer này sử dụng một cửa sổ trượt quét qua toàn bộ ảnh dữ liệu, mỗi lần trượt theo một bước trượt (stride) cho trước. Khác với layer Convolution, layer Pooling không tính tích chập mà tiến hành lấy mẫu (subsampling). Khi cửa sổ trượt trên ảnh, chỉ có một giá trị được xem là giá trị đại diện cho thông tin ảnh tại vùng đó (giá trị mẫu) được giữ lại. Các phương thức lấy phổ biến trong layer Pooling là MaxPooling (lấy giá trị lớn nhất), MinPooling (lấy giá trị nhỏ nhất) và AveragePooling (lấy giá trị trung bình).

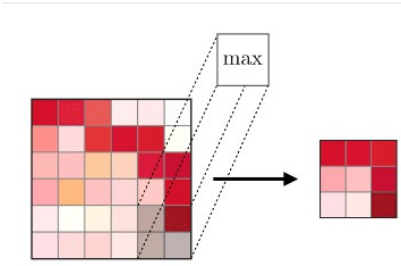
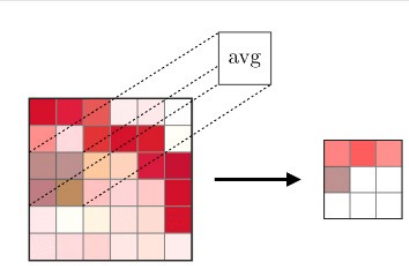
Xét một ảnh có kích thước 32×32 và layer Pooling sử dụng bộ lọc có kích thước 2×2 với bước trượt stride là 2, phương pháp sử dụng là MaxPooling. Bộ lọc sẽ lần lượt trượt qua ảnh, với mỗi lần trượt chỉ có giá trị lớn nhất trong 4 giá trị nằm trong vùng cửa sổ 2×2 của bộ lọc được giữ lại và đưa vào ma trận đầu ra. Như vậy, sau khi qua layer Pooling, ảnh sẽ giảm kích thước xuống còn 16×16 (kích thước mỗi chiều giảm 2 lần).



Hình 1.36. Hình ảnh minh họa Max Pooling

Pooling Layer có vai trò giảm kích thước dữ liệu. Với một bức ảnh kích thước lớn qua nhiều Pooling Layer sẽ được thu nhỏ lại tuy nhiên vẫn giữ được những đặc trưng cần cho việc nhận dạng (thông qua cách lấy mẫu). Việc giảm kích thước dữ liệu sẽ làm giảm lượng tham số, tăng hiệu quả tính toán và góp phần kiểm soát hiện tượng quá khớp (overfitting).

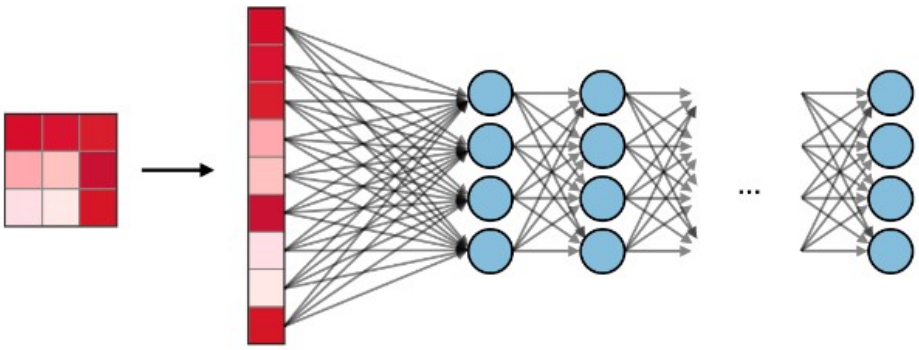
Có nhiều dạng pooling cho các bài toán cụ thể, dưới đây em xin giới thiệu về hai dạng pooling hay được sử dụng nhất đó là max pooling và average pooling:

Kiểu	**Max pooling**	**Average pooling**
Chức năng	Từng phép pooling chọn giá trị lớn nhất trong khu vực mà nó đang được áp dụng	Từng phép pooling tính trung bình các giá trị trong khu vực mà nó đang được áp dụng
Minh họa		
Nhận xét	• Bảo toàn các đặc trưng • Được sử dụng nhiều	• Giảm kích thước feature map • Được sử dụng trong mạng LeNet

Bảng 1-1. So sánh Max Pooling và Average Pooling [10]

❖ *Lớp kết nối đầy đủ*

Tầng kết nối đầy đủ (FC) nhận đầu vào là các dữ liệu đã được làm phẳng, mà mỗi đầu vào đó được kết nối đến tất cả neuron. Trong mô hình mạng CNNs, các tầng kết nối đầy đủ thường được tìm thấy ở cuối mạng và được dùng để tối ưu hóa mục tiêu của mạng ví dụ như độ chính xác của lớp.



Hình 1.37. Minh họa lớp fully connected [10]

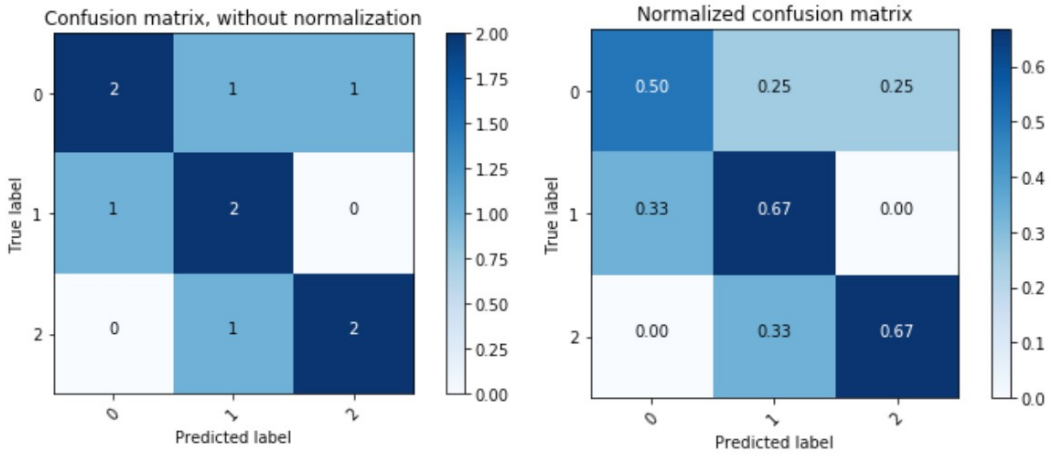
1.7. Chỉ số đánh giá mô hình

Đối với bài toán sử dụng mô hình học sâu (Deep Learning) hay các thuật toán học máy (Machine Learning) nói chung luôn luôn phải có phương thức để đánh giá sự hiệu quả của mô hình. Từ các kết quả đánh giá mô hình đó ta có thể trích rút ra được mô hình phù hợp với bài toán thực tại để áp dụng vào công đoạn xây dựng hệ thống phần mềm. Chính vì vậy, để có một cái nhìn về thước đo đánh giá cụ thể em sẽ định nghĩa các thước đo quan trọng và được ứng dụng trong quá trình đánh giá mô hình của đề tài này.

1.7.1. Ma trận đánh giá Confusion matrix

Confusion matrix không được coi là một chỉ số đánh giá mô hình nhưng nó lại mang đến cái nhìn tổng quát cho các nhà khoa học khi đánh giá hiệu năng của một mô hình. Đối với các bài toán phân loại, confusion matrix thể hiện được có bao nhiêu điểm dữ liệu thực sự thuộc vào một class và được dự đoán rơi vào một class. Điều này có nghĩa là confusion matrix sẽ cho ta thấy các kết quả tổng quan nhất khi một mô hình dự đoán các nhãn của đề tài có sẵn. Cách biểu diễn của confusion matrix sẽ có người xem biết được số lượng giá trị mà mô hình dự đoán sai hay đúng trên một nhãn thực tế.

Ví dụ hình ảnh dưới đây sẽ cho ta thấy cái nhìn rõ ràng nhất về confusion matrix này:



Hình 1.38. Minh họa unnormalized confusion matrix và confusion matrix [11]

Đối với 2 hình ảnh trên, ta đều quy ước trục hoàng là nhãn mà mô hình dự đoán (predicted label) và trục tung là nhãn thực tế của dữ liệu (true label). Sau đây ta sẽ đi phân tích chi tiết vào hai hình minh họa trên hình 37 như sau:

- Unnormalized confusion matrix (Hình bên trái): Đối với mỗi ô đều thể hiện số lượng dữ liệu thực tế được mô hình dự đoán. Ví dụ đối với nhãn thực tế là 0, ta có thể thấy rằng mô hình dự đoán chính xác nhãn bằng 0 trên 2 dữ liệu (ô ở vị trí [0][0]), dự đoán sai nhãn bằng 1 trên 1 dữ liệu (ô ở vị trí [0][1]) và dự đoán sai nhãn bằng 2 trên 1 dữ liệu (ô ở vị trí [0][2]). Tương tự đối với các dữ liệu có nhãn thực tế bằng 1 và 2 cũng như vậy. Như vậy unnormalized confusion matrix cho ta thấy số lượng mà mô hình dự đoán đối với từng nhãn cụ thể.
- Confusion matrix (Hình bên phải): Khác với unnormalized confusion matrix thì confusion matrix lại thể hiện xác suất dự đoán của từng nhãn. Ví dụ như đối với nhãn thực tế là 0 thì mô hình đã dự đoán chính xác tới 0.5 (ô ở vị trí [0][0]) và dự đoán sai với hai nhãn còn lại tỉ lệ là 0.25 (lần lượt các ô [0][1] và [0][2]). Như vậy, đối với confusion matrix ta có thể nhận được kết quả là tỉ lệ mà mô hình dự đoán từng nhãn. Điều này sẽ giúp các nhà khoa học có thể nhận ra rằng mô hình đang có thiên hướng dự đoán như thế nào để tiến hành điều chỉnh trong tương lai.

Tóm lại, đối với confusion matrix chúng ta nhận được cái nhìn rõ ràng nhất về kết quả dự đoán của mô hình đối với từng nhãn. Điều này mang lại rất nhiều lợi ích cho các nhà khoa học trong việc điều chỉnh mô hình và quan sát xu hướng dự đoán của mô hình trong tương lai.

1.7.2. Độ chính xác accuracy

Accuracy là độ đo bài toán phân loại đơn giản nhất vì nó chỉ lấy tổng số dự đoán đúng chia cho toàn bộ các dự đoán. Hay nói cách khác nếu đối chiếu với unnormalized confusion matrix được giới thiệu ở trên thì nó là tỉ lệ tổng đường chéo chính trên tổng các ô của unnormalized confusion matrix.

Ưu điểm của chỉ số đánh giá này là cho ta thấy được tỉ lệ chính xác của việc mô hình dự đoán và vô cùng dễ dàng tính toán. Tuy nhiên, nhược điểm của cách đánh giá này chỉ cho ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất hay dữ liệu của lớp nào thường bị phân loại nhầm nhất vào các lớp khác. Ngoài ra, chỉ số đánh giá này chỉ nên áp dụng đối với các bài toán có lượng dữ liệu các nhãn cân bằng nhau vì nếu số lượng các nhãn có một bên chênh lệch quá nhiều, nếu mô hình chỉ luôn luôn cho kết quả về nhãn của dữ liệu đó cũng có thể đem lại kết quả cao. Ví dụ bài toán dự đoán chó – mèo, nếu lượng nhãn dữ liệu về chó trong thực tế chiếm đến hơn 90% thì chỉ cần mô hình luôn luôn dự đoán là chó cũng sẽ đem lại accuracy cao mà thực tế đó là một mô hình không hiệu quả.

1.7.3. Precision

Như đã nói ở trên, sẽ có rất nhiều trường hợp thước đo accuracy không phản ánh đúng hiệu quả của mô hình nhất là đối với trường hợp tập dữ liệu có số lượng nhãn chênh lệch lớn. Vì vậy cần có chỉ số để khắc phục yếu điểm của accuracy. Chính vì vậy, precision là một trong những chỉ số khắc phục được điều này và nó được thể hiện qua công thức dưới đây:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Ở đây, ta thấy xuất hiện các thuật ngữ mới bao gồm: "True Positive, False Positive". Thực chất hai thuật ngữ này chỉ đơn giản là đại diện cho việc dự đoán đúng với nhãn tích cực (True Positive) và dự đoán sai với nhãn tích cực (False Positive). Nhãn tích cực (positive) ở đây thực chất là việc đặt tên trong bài toán phân loại nhị phân, được chia làm hai loại nhãn tích cực (positive) và nhãn tiêu cực (negative). Chỉ số precision ở đây về bản chất là đánh giá tỉ lệ nhãn dự positive trong thực tế trên toàn bộ dữ liệu được phân loại là positive. Điều này có nghĩa là chỉ số precision để đánh giá xem rằng tỉ lệ các nhãn đúng là positive trên các nhãn được dự đoán là positive và nếu chỉ số này cao đồng nghĩa rằng độ chính xác các điểm tìm được là cao.

1.7.4. Recall

Recall cũng là một metric quan trọng, nó đo lường tỷ lệ dự báo chính xác các trường hợp positive trên toàn bộ các mẫu thuộc nhóm positive. Recall cao đồng nghĩa với việc True Positive Rate cao, tức là tỷ lệ bỏ sót các điểm thực sự là positive là thấp. Công thức của Recall như sau:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Vậy thì rõ ràng ở đây có sự khác nhau giữa hai chỉ số precision và recall. Nếu precision là tỉ lệ nhãn thực tế là positive trên toàn bộ các nhãn được dự đoán là positive thì recall lại là tỉ lệ các trường hợp positive trên toàn bộ dữ liệu thực tế là positive. Trong thực tế, đối với bài toán chuẩn đoán bệnh ung thư, chúng ta sẽ tập trung vào việc bắt hết các bệnh nhân thực tế bị ung thư cho dù có nhầm với người bình thường vì còn hơn là bỏ xót nên việc sử dụng chỉ số recall là cần thiết đối với trường hợp này.

1.7.5. F1 score

Tùy thuộc vào bài toán mà bạn sẽ muốn ưu tiên sử dụng Recall hay Precision. Nhưng cũng có rất nhiều bài toán mà cả Precision hay Recall đều quan trọng. Một metric phổ biến đã kết hợp cả Recall và Precision lại được gọi là F1-score. F1-score được tính theo công thức sau:

$$F1-score=\frac{2\times Precision\times Recall}{Precision+Recall}$$

Trên đây, là những chỉ số để đánh giá đối với các bài toán phân loại hình ảnh. Tiếp theo em sẽ trình bày thêm về các chỉ số đánh giá đối với các bài toán xử lý ảnh để so sánh hiệu năng sinh ra ảnh mới với kết quả ảnh thực tế.

1.7.6. Structural Similarity Index Measurement (SSIM)

Chỉ số SSIM được sử dụng để đo mức độ giống nhau giữa hình ảnh đầu vào và ảnh sinh ra. Công thức SSIM dựa trên ba thông số để so sánh bao gồm: độ chói (luminance), tương phản (contrast) và cấu trúc (structure). [12] Một ảnh so với ảnh gốc là tốt nếu:

- Những điểm ảnh có mức độ sáng tối khác nhau, và càng có nhiều mức độ sáng tối càng có nhiều chi tiết ảnh => Ảnh chất lượng tốt
- Một bức ảnh không phải độ tương phản càng cao thì càng tốt mà nên có sự hài hòa cân đối giữa sáng và tối. => Độ đa dạng

Từ đó ta có công thức đề xuất:

$$SSIM(x,y)=[l(x,y)]^{\alpha}\times[c(x,y)]^{\beta}\times[s(x,y)]^{\gamma}$$

Giá trị SSIM sẽ xuất hiện trong khoảng -1 đến 1, đạt giá trị bằng 1 trong trường hợp hai dữ liệu ảnh giống hệt nhau. Như vậy, chỉ số này có giá trị càng lớn thì tương ứng với model càng tốt.

1.7.7. PSNR

PSNR được định nghĩa thông qua sai số toàn phương trung bình (MSE – Mean squared error). MSE là một khái niệm trong thống kê học, nghĩa là sai số toàn phương trung bình của một phép ước lượng là trung bình của bình phương các sai số, nghĩa là sự khác biệt giữa các ước lượng và những gì đánh giá. [12] Ở

đây MSE được xác định cho ảnh hai chiều có kích thước mxn trong đó I và K là ảnh gốc và ảnh sau khi tổng hợp.

$$MSE = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

$$PSNR = 10.\log_{10} \frac{MAX_I^2}{MSE} = 20.\log_{10} (\frac{MAX_I^2}{\sqrt{MSE}})$$

Ở đây MAX_I là giá trị tối đa của pixel trên ảnh. Khi các pixels được biểu diễn bởi 8 bits, thì giá trị của nó là 255. Trường hợp tổng quát khi tín hiệu được biểu diễn bởi B bit trên một đơn vị mẫu MAX_I là 2^{B-1}.

Thông thường nếu PSNR>=40 dB thì hệ thống mắt thường gần như không phân biệt được ảnh gốc và ảnh sinh ra. PSNR càng cao thì chất lượng ảnh sinh càng tốt, khi 2 ảnh giống hệt nhau thì MSE=0 và PSNR đi đến vô hạn, đơn vị của PSNR là Decibel.

❖ *Kết luận chương*

Như vậy, nội dung chương 1 đã giới thiệu chi tiết về hiện trạng thực tế của đồ án, các mục tiêu và đối tượng mà đồ án muốn giải quyết. Bên cạnh đó, chương 1 cũng giới thiệu những định hướng giải pháp cơ bản để xây dựng đồ án để từ đó đi tới các lý thuyết quan trọng như Deep Learning và phân tích mạng nơ – ron nhân tạo. Những nội dung lý thuyết về mạng nơ – ron nhân tạo, Deep Learning đó phần nào thể hiện cái nhìn tổng quan về phương thức vận hành và ứng dụng của các mạng nơ-ron nhân tạo hay các mô hình học sâu. Ngoài ra, chương 1 cũng giới thiệu chi tiết các chỉ số đánh giá, giải thích cách tính toán, phân tích trường hợp sử dụng để hiểu biết thêm về những thước đo đánh giá độ hiệu quả của một mô hình. Tiếp theo là chương 2, đồ án sẽ giới thiệu sơ đồ hoạt động tổng quát và luồng vận hành các chức năng của chương trình. Bên cạnh đó, chương 2 sẽ trình bày chi tiết đến các thành phần được sử dụng để xây dựng mô hình tới người đọc.

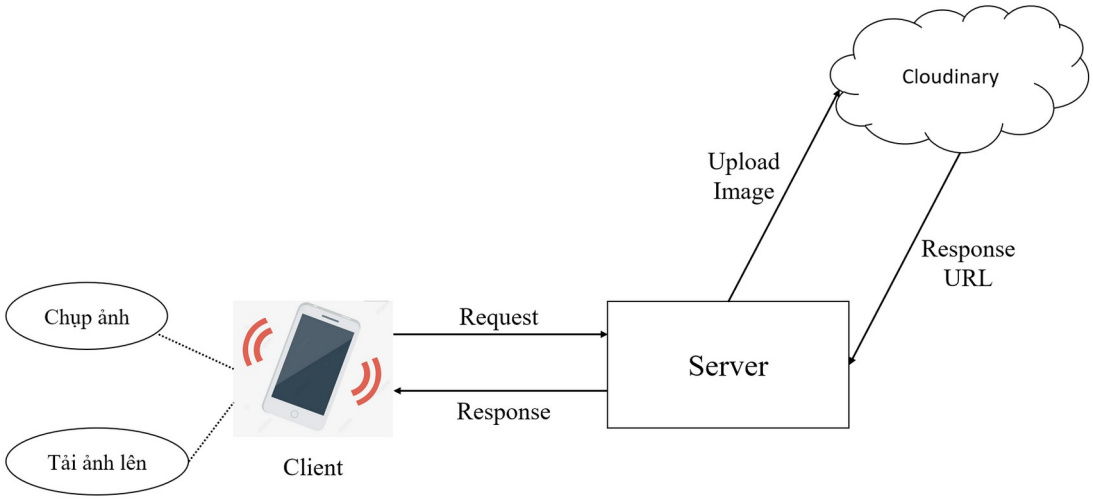
CHƯƠNG 2. CÁC THÀNH PHẦN HỆ THỐNG

Nội dung chương 2 sẽ thể hiện cái nhìn tổng quan về hệ thống và luồng vận hành đối với hai chức năng phát hiện ảnh mờ và khôi phục ảnh mờ của chương trình. Bên cạnh đó chương 2 sẽ trình bày về các thành phần cấu tạo lên toàn bộ chương trình, các lý thuyết liên quan đến việc xây dựng mô hình cho chương trình. Chương 2 sẽ bao gồm tuần tự các thành phần sau:

- Giới thiệu tổng quan về sơ đồ hệ thống và luồng vận hành của các chức năng
- Trình bày các công cụ xây dựng lên chương trình phần mềm
- Trình bày lý thuyết về các thuật toán, mô hình học máy và mô hình học sâu sẽ được thử nghiệm trong đồ án

2.1. Tổng quan về hệ thống

2.1.1. Sơ đồ hoạt động hệ thống



Hình 2.39. Hình ảnh minh họa sờ đồ hoạt động hệ thống

Hệ thống phần mềm "Phát hiện và khôi phục ảnh mờ" được phát triển để cung cấp khả năng tải ảnh từ thiết bị di động lên một server để xử lý, sau đó đẩy ảnh lên dịch vụ lưu trữ đám mây Cloudinary. Quá trình này giúp quản lý hiệu suất và quản lý tài nguyên ảnh một cách hiệu quả, đồng thời cung cấp URL trả về cho client. Hệ thống phần mềm bao gồm hai chức năng chính đó là phân loại ảnh mờ hay nét và khử mờ tấm ảnh đó để trả về kết quả.

❖ *Kiến trúc hệ thống cơ bản:*

- ***Client (Ứng Dụng Android):*** Người dùng sử dụng ứng dụng Android để chọn và tải ảnh lên server.
- ***Server (Django Rest Framework)****:* Nhận và xử lý ảnh từ client, sau đó tương tác với dịch vụ Cloudinary.

- ***Cloudinary:*** Dịch vụ lưu trữ đám mây chịu trách nhiệm lưu trữ và quản lý ảnh theo cách an toàn và linh hoạt.

❖ *Luồng hoạt động cơ bản*

- Bước 1: Tải Ảnh Lên Server
  - ✓ Người dùng chọn ảnh từ thiết bị Android.
  - ✓ Client gửi yêu cầu tải ảnh lên server.
  - ✓ Server nhận và lưu trữ ảnh vào bộ nhớ tạm thời.
- Bước 2: Xử Lý Ảnh Trên Server
  - ✓ Server thực hiện các xử lý cần thiết trên ảnh, cụ thể: phân loại hình ảnh, phân vùng mờ và khôi phục ảnh
  - ✓ Ảnh được lưu trữ tạm thời hoặc có thể lưu trực tiếp vào Cloudinary nếu cần.
- Bước 3: Đẩy Ảnh Lên Cloudinary
  - ✓ Server tương tác với API của Cloudinary để đẩy ảnh lên dịch vụ.
  - ✓ Dịch vụ Cloudinary nhận và lưu trữ ảnh theo cách an toàn và có thể tùy chỉnh.
- Bước 4: Trả Về URL Cho Client
  - ✓ Sau khi ảnh được thành công đẩy lên Cloudinary, server trả về URL của ảnh cho client.
  - ✓ Client có thể sử dụng URL này để hiển thị hoặc chia sẻ ảnh.

Hệ thống phần mềm Android này cung cấp một quy trình mạnh mẽ và linh hoạt để tải ảnh lên server, xử lý chúng và lưu trữ an toàn trên đám mây. Việc sử dụng Cloudinary giúp quản lý tài nguyên ảnh một cách hiệu quả, đồng thời cung cấp trải nghiệm người dùng tốt nhất.

2.1.2. Luồng hoạt động các chức năng hệ thống

❖ *Chức năng phát hiện ảnh mờ*

Chức năng phát hiện ảnh mờ của hệ thống có nhiệm vụ nhận đầu vào là một hình ảnh được chụp hoặc tải lên từ thiết bị di động và trả về kết quả dự đoán đó là một tấm ảnh mờ hay không. Dưới đây là các bước thực hiện chi tiết chức năng phát hiện ảnh mờ của hệ thống:

- Bước 1: Ảnh được chụp hoặc tải lên từ thiết bị di động gửi về server.
- Bước 2: Server nhận hình ảnh và xử lý hình ảnh bằng thuật toán.
- Bước 3: Server đánh dấu các vùng mờ bằng các pixel màu trắng và các vùng không mở bằng các pixel màu đen.
- Bước 4: Server đẩy ảnh sau xử lý lên cloudinary và nhận về url chứa hình ảnh đã được xử lý.
- Bước 5: Server phản hồi kết quả dự đoán ảnh mờ và đường dẫn ảnh sau xử lý.

- Bước 6: Client tiếp nhận phản hồi từ server và hiển thị kết quả cùng với ảnh phân vùng mờ đã xử lý

❖ *Chức năng khôi phục ảnh mờ*

Chức năng khôi phục ảnh mờ của hệ thống có nhiệm vụ nhận đầu vào tiếp tục là tấm ảnh vừa được dự đoán ở chức năng trên và trả về kết quả là một hình ảnh mới đã được khôi phục. Dưới đây là các bước thực hiện chi tiết chức năng khôi phục ảnh mờ của hệ thống:

- Bước 1: Người dùng tiếp tục gửi hình ảnh đã xử lý lên server.
- Bước 2: Server nhận được hình ảnh muốn khử mờ và tiến hành chạy mô hình khử mờ.
- Bước 3: Server khử mờ thành công và tiếp tục đẩy ảnh đã khử mờ lên cloudinary.
- Bước 4: Cloudinary nhận ảnh và lưu trữ ảnh rồi trả về một đường dẫn url của ảnh.
- Bước 5: Server nhận đường dẫn ảnh đó và trả về phía client.
- Bước 6: Ứng dụng nhận được đường dẫn ảnh và hiển thị kết quả ảnh đã được khử mờ.

2.2. Các công cụ lập trình hệ thống

2.2.1. Công cụ lập trình phía Client

❖ *Ngôn ngữ lập trình Kotlin*

Kotlin là một ngôn ngữ lập trình mới, đa nền tảng (cross-platform), và chủ yếu được phát triển bởi JetBrains. Được công bố công khai vào năm 2011, Kotlin đã nhanh chóng trở thành một trong những ngôn ngữ lập trình phổ biến trên nền tảng Java, đặc biệt là khi nó được chọn làm ngôn ngữ chính thức hỗ trợ cho phát triển ứng dụng Android.

Hình 2.40. Hình ảnh minh họa Kotlin

Dưới đây có thể kể đến các điểm mạnh của ngôn ngữ lập trình Kotlin có thể giúp nó trở thành một sự lựa chọn hấp dẫn cho các dự án phần mềm như sau:

- ***Tương thích ngược với Java:*** Kotlin được thiết kế để tương thích ngược với mã nguồn Java. Điều này có nghĩa là bạn có thể tích hợp mã nguồn Kotlin vào dự án Java hiện tại mà không gặp vấn đề tương thích lớn.
- ***An toàn và Tính năng loại dữ liệu:*** Kotlin hỗ trợ loại dữ liệu nullable, giúp giảm thiểu lỗi null pointer exceptions, một vấn đề phổ biến trong Java.
- ***Concise và Đọc ghi mã dễ hiểu:*** Mã nguồn Kotlin thường ngắn gọn hơn so với Java, giúp giảm độ phức tạp và làm cho mã nguồn trở nên dễ đọc, dễ hiểu hơn.
- ***Hỗ trợ mạnh mẽ cho Extension Functions:*** Kotlin có khái niệm về extension functions, giúp mở rộng chức năng của các lớp mà không cần thay đổi mã nguồn nguyên.
- ***Lập trình hàm (Functional Programming):*** Kotlin hỗ trợ lập trình hàm với lambda expressions, higher-order functions, và các tính năng khác, giúp viết mã ngắn gọn và linh hoạt.
- ***Cộng đồng sôi động:*** Kotlin có một cộng đồng người dùng đang ngày càng phát triển và tích cực đóng góp vào việc phát triển ngôn ngữ này.
- ***Phát triển ứng dụng Android:*** Kotlin đã được chọn làm ngôn ngữ chính thức hỗ trợ cho phát triển ứng dụng Android, thay thế cho Java. Điều này mang lại nhiều lợi ích cho những nhà phát triển Android, bao gồm hiệu suất cải thiện và tính năng ngôn ngữ tiên tiến.
- ***Đa nền tảng (Multiplatform):*** Kotlin cung cấp khả năng phát triển ứng dụng đa nền tảng, giúp chia sẻ mã nguồn giữa các nền tảng như JVM, Android, JavaScript, và Native.

Trong đề tài đồ án lần này, Kotlin được sử dụng kết hợp với việc áp dụng các mô hình thử nghiệm và đánh giá để xây dựng một ứng dụng Android cơ bản nhằm đưa ra các kết quả thực tế.

❖ *Thư viện Gson*

Gson là một thư viện mã nguồn mở của Google, được sử dụng để chuyển đổi giữa đối tượng Java và định dạng JSON. Gson cung cấp các phương thức đơn giản và linh hoạt để thực hiện việc serialize (chuyển đối tượng Java thành JSON) và deserialize (chuyển JSON thành đối tượng Java).

Hình 2.41. Hình ảnh minh họa Gson chuyển đổi dữ liệu

Được thiết kế để tích hợp một cách dễ dàng và hiệu quả vào ứng dụng Java, Gson tự động xử lý quá trình ánh xạ giữa cấu trúc dữ liệu trong Java và cấu trúc dữ liệu JSON tương ứng mà không cần phải viết mã chuyển đổi tự động. Gson hỗ trợ các tính năng như:

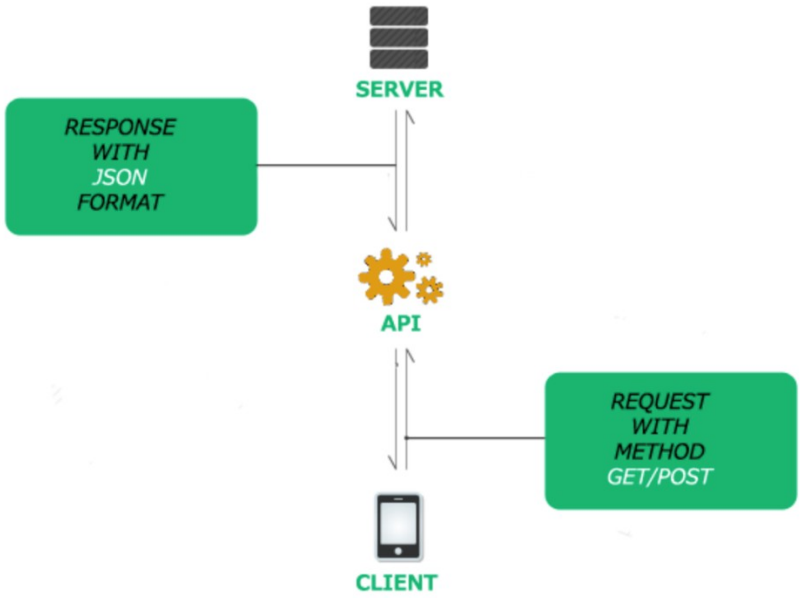
- ***Deserialize tự động**:* Gson có thể tự động ánh xạ dữ liệu từ JSON vào các đối tượng Java một cách linh hoạt và thuận tiện.
- ***Serialize tự động:*** Chuyển đổi tượng Java thành định dạng JSON một cách dễ dàng và tự động.
- ***Hỗ trợ generics:*** Gson hỗ trợ đối với các kiểu dữ liệu generics, giúp xử lý các cấu trúc dữ liệu phức tạp.
- ***Customization:*** Cho phép người dùng tùy chỉnh quá trình serialize và deserialize thông qua sự mở rộng của Gson Adapter.

❖ *Thư viện Retrofit*

Retrofit là một thư viện mã nguồn mở của Square được sử dụng trong phát triển ứng dụng Android và Java để tạo và quản lý các yêu cầu mạng (network requests). Thư viện này giúp đơn giản hóa việc tương tác với các API web bằng cách chuyển đổi các yêu cầu HTTP thành các gọi hàm trực tiếp trong mã nguồn, làm giảm sự phức tạp của việc xử lý mã nguồn khi làm việc với RESTful API. Dưới đây là một số đặc điểm và ưu điểm chính của Retrofit:

- ***Annotation-based API Definition:*** Retrofit sử dụng các chú thích (annotations) để định nghĩa các yêu cầu mạng trên các phương thức Java, giúp giảm đi lượng mã boilerplate và làm tăng sự đơn giản và đọc ghi của mã nguồn.
- ***Converter System:*** Retrofit hỗ trợ nhiều loại chuyển đổi dữ liệu (converter), cho phép bạn chuyển đổi dữ liệu giữa định dạng JSON, XML và các định dạng khác dễ dàng.

- ***OkHttp Integration:*** Retrofit tích hợp chặt chẽ với thư viện OkHttp để quản lý yêu cầu và phản hồi mạng. Điều này cung cấp khả năng tùy chỉnh, theo dõi và gỡ lỗi tốt hơn.
- ***Asynchronous and Synchronous Execution:*** Retrofit hỗ trợ cả thực hiện đồng bộ (synchronous) và đồng bất đồng bộ (asynchronous), giúp ứng dụng có thể thực hiện các yêu cầu mạng mà không làm đóng băng giao diện người dùng.
- ***Dynamic URL Support:*** Retrofit cho phép bạn thực hiện các yêu cầu mạng đến các URL được xác định động tại thời điểm chạy, giúp linh hoạt khi làm việc với các API có các phần của URL thay đổi.



Hình 2.42. Minh họa Retrofit

2.2.2. Công cụ lập trình phía Server

❖ *Django Rest Framework*

Django Rest Framework (DRF) là một framework mạnh mẽ được xây dựng trên nền tảng Django, được thiết kế đặc biệt để phát triển các API web RESTful trong ứng dụng Django. DRF cung cấp một loạt các tính năng và công cụ để giúp lập trình viên dễ dàng xây dựng và quản lý các API một cách hiệu quả. Dưới đây là một số điểm nổi bật về Django Rest Framework:

- ***Serialization:*** DRF cung cấp mô hình serialization mạnh mẽ để chuyển đổi đối tượng Django (như models và queryset) thành định dạng dữ liệu phổ biến như JSON. Ngược lại, nó cũng hỗ trợ chuyển đổi dữ liệu từ định dạng JSON thành các đối tượng Django.
- ***View Classes:*** DRF cung cấp các lớp view như APIView và ViewSet để xử lý yêu cầu HTTP và tương tác với dữ liệu. Các lớp này hỗ trợ nhiều phương thức HTTP như GET, POST, PUT, DELETE.

- ***Authentication and Permissions**:* DRF có các lớp xác thực (authentication) và kiểm soát quyền (permissions) để quản lý quyền truy cập vào các API. Bạn có thể dễ dàng xác định ai được phép thực hiện những hành động như đọc, ghi, cập nhật, xóa dữ liệu.
- ***Pagination:*** DRF hỗ trợ phân trang (pagination) tự động cho các kết quả trả về từ các API, giúp kiểm soát lượng dữ liệu được trả về trong mỗi yêu cầu.
- ***DRF Router:*** Router của DRF giúp xác định các URL tự động dựa trên các action trong viewset, giảm độ phức tạp của việc quản lý URL patterns.
- ***Throttling:*** Để ngăn chặn tấn công DDoS, DRF hỗ trợ cơ chế throttling để kiểm soát tần suất của yêu cầu từ một nguồn cụ thể.
- ***Đa nền tảng*:** DRF không chỉ hỗ trợ Django truyền thống mà còn có thể tích hợp dễ dàng với các ứng dụng Django RESTful.

Đối với đề tài này, em sẽ sử dụng Django Rest Framework để xây dựng server đảm nhiệm các công việc xử lý ảnh và trả anh về cho client

Hình 2.43. Minh họa Django Rest Framework

❖ *Cloudinary*

Cloudinary là một dịch vụ quản lý và tối ưu hóa tài nguyên đa phương tiện trực tuyến, chủ yếu tập trung vào quản lý hình ảnh và video. Dịch vụ này cung cấp các công cụ mạnh mẽ để tải lên, lưu trữ, xử lý và phục vụ các tài nguyên đa phương tiện, giúp các nhà phát triển và doanh nghiệp tối ưu hóa trải nghiệm người dùng của họ. Các điểm nổi bật của Cloudinary bao gồm:

- ***Quản lý hình ảnh và video*:** Cloudinary giúp quản lý và tối ưu hóa hình ảnh và video một cách hiệu quả. Dịch vụ này cung cấp các API và công cụ để chuyển đổi, cắt ghép, tối ưu hóa kích thước, và thậm chí thực hiện các hiệu ứng trên hình ảnh và video.

- ***Lưu trữ đám mây:*** Cloudinary cung cấp một giải pháp lưu trữ đám mây, giúp giảm áp lực lưu trữ trên máy chủ của bạn và tăng tính sẵn có của tài nguyên đa phương tiện.
- ***Phục vụ nhanh và CDNs tích hợp:*** Cloudinary sử dụng Content Delivery Network (CDN) để phục vụ tài nguyên một cách nhanh chóng và hiệu quả. Điều này giúp cải thiện thời gian tải trang và trải nghiệm người dùng.
- ***Tùy chỉnh và tích hợp dễ dàng:*** Cloudinary hỗ trợ nhiều ngôn ngữ lập trình và framework, cũng như tích hợp với nhiều dịch vụ lưu trữ khác nhau như Amazon S3, Google Cloud Storage, Dropbox, và nhiều dịch vụ đám mây khác.
- ***Quản lý tài nguyên thông minh:*** Dịch vụ này có khả năng tự động quản lý và tối ưu hóa tài nguyên đa phương tiện dựa trên nền tảng máy học, giúp tiết kiệm băng thông và tăng hiệu suất.
- ***Bảo mật và quyền riêng tư:*** Cloudinary cung cấp các tính năng bảo mật như quản lý quyền truy cập, chứng nhận và mã hóa để đảm bảo an toàn cho dữ liệu và tài nguyên của người dùng.

Hình 2.44. Hình ảnh minh họa cloudinary

Cloudinary là một công cụ mạnh mẽ cho việc quản lý tài nguyên đa phương tiện trực tuyến, và nó được sử dụng rộng rãi trong các ứng dụng web và di động để cải thiện hiệu suất và trải nghiệm người dùng. Trong đề tài, em cũng sử dụng cloudinary để lưu trữ những hình ảnh được xử lý và sinh ra url để phía client có thể nhận để hiển thị hình ảnh được xử lý.

2.3. Các mô hình và thuật toán thử nghiệm

2.3.1. Mô hình học máy (Machine Learning) truyền thống

Một trong những công việc nổi bật của đề tài đồ án này là việc tiến hành thử nghiệm các mô hình học máy truyền thống Machine Learning cho

tác vụ phát hiện ảnh mờ như thuật toán K – Nearest Neighbors, thuật toán Naïve Bayes, thuật toán Decision Tree, thuật toán Random Forest. Các thuật toán được thử nghiệm với nhiều kiểu điều chỉnh tham số và được thực hiện trên thư viện scikit-learn của Python. Sau đây là nội dung giới thiệu sơ lược về các thuật toán được thử nghiệm:

❖ *Thuật toán K- Nearest Neighbors*

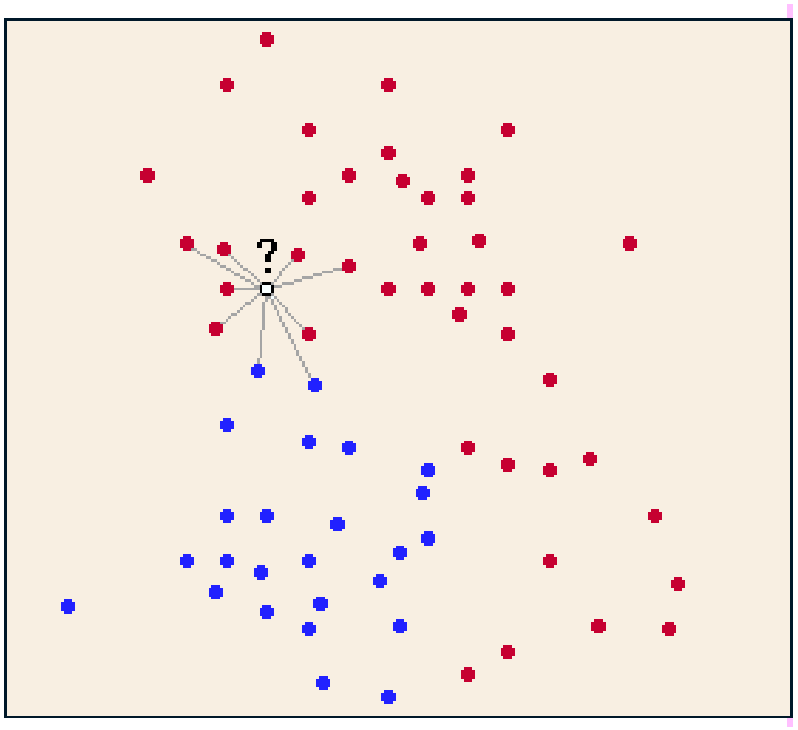
Thuật toán K-Nearest Neighbors (KNN) là một thuật toán học máy đơn giản và phổ biến được sử dụng trong cả phân loại (classification) và hồi quy (regression). KNN là một thuật toán dựa trên việc đo đạc sự tương đồng giữa các điểm dữ liệu và dựa vào đó để đưa ra dự đoán hoặc phân loại. Thuật toán này còn có tên gọi khác là học lười vì gần như nó không phải học tập trên tập dữ liệu như các thuật toán khác mà nó sẽ dựa vào các điểm dữ liệu gần nó.

Nguyên lý hoạt động của KNN hoạt động dựa trên giả định rằng các điểm dữ liệu có thuộc tính tương tự sẽ nằm gần nhau trong không gian đặc trưng. Khi có một điểm dữ liệu mới cần phân loại hoặc dự đoán, KNN tìm K điểm dữ liệu gần nhất trong không gian đặc trưng và sử dụng đa số phiếu bầu để xác định nhãn hoặc giá trị của điểm đó.

Bởi vì việc dự đoán kết quả của KNN dựa trên các điểm dữ liệu gần với nó cho nên ta cần phải có một công thức cho thước đo khoảng cách giữa các điểm dữ liệu. KNN thường sử dụng các phương pháp đo đạc khoảng cách như Euclidean distance, Manhattan distance, hoặc các phương pháp đo đạc khác tùy thuộc vào bối cảnh của vấn đề.

Giá trị K là số lượng hàng xóm gần nhất mà thuật toán KNN sẽ xem xét khi thực hiện dự đoán. Lựa chọn giá trị K đòi hỏi sự cân nhắc, vì giá trị này có thể ảnh hưởng đến hiệu suất của mô hình.

Bởi vì tính đơn giản của thuật toán cho nên KNN thường được sử dụng hiệu quả trên các bộ dữ liệu có số chiều thấp (low-dimensional), vì khi số chiều tăng lên, việc đo đạc khoảng cách giữa các điểm dữ liệu trở nên phức tạp.



Hình 2.45. Hình ảnh minh hoạt thuật toán K - Nearest Neighbors

KNN có thể chịu được nhiễu và không cần huấn luyện trước, nhưng nó cũng có thể nhạy cảm với nhiễu và outliers tùy thuộc vào dữ liệu láng giềng. Thuật toán này thường được sử dụng trong các ứng dụng như phân loại hình ảnh hay bất kỳ vấn đề nào mà phụ thuộc vào việc đo đạc sự tương đồng giữa các điểm dữ liệu.

Đối với các thử nghiệm lần này của đồ án tốt nghiệp sẽ áp dụng mô hình KNN dựa trên thư viện scikit-learn của Python với đa dạng các tham số K khác nhau ứng với lần lượt các giá trị như 12,13,14,15,16,17,18.

❖ *Thuật toán Naïve Bayes*

Naive Bayes là một thuật toán phân loại (classification) dựa trên nguyên tắc của định lý Bayes. Thuật toán này giả định rằng các đặc trưng đầu vào độc lập với nhau, điều này là một giả định mạnh mẽ nhưng giúp giảm độ phức tạp của mô hình.

Naive Bayes sử dụng định lý Bayes để tính xác suất của một lớp dựa trên các đặc trưng đầu vào. Công thức cơ bản của thuật toán này như sau:

$$P(class/data)=\frac{P(data/class)\times P(class)}{P(data)}$$

Việc giả định các đặc trưng đầu vào là độc lập nhau sẽ giảm bớt độ phức tạp của mô hình cho phép việc tính xác suất của mỗi đặc trưng độc lập và có thể nhân chúng lại. Mô hình được huấn luyện với một tập dữ liệu đã được gán nhãn, trong đó tính toán xác suất tiên nghiệm của các lớp và xác suất của mỗi đặc trưng dựa trên lớp. Khi có một điểm dữ liệu mới, mô hình tính toán xác suất hậu

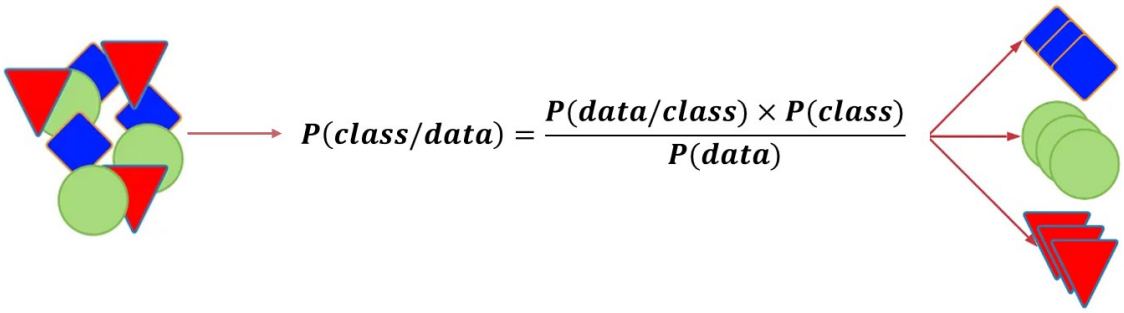
nghiệm của từng lớp và chọn lớp có xác suất cao nhất. Các ứng dụng thực tế của mô hình có thể kể tới như

- ***Phân Loại Văn Bản:*** Naive Bayes thường được sử dụng trong phân loại văn bản, như phân loại thư rác và không thư rác.
- ***Dự Đoán Sự Kiện:*** Áp dụng trong dự đoán sự kiện dựa trên các đặc trưng như thời tiết, điều kiện kinh tế, v.v.
- ***Hệ Thống Phân Loại:*** Sử dụng để phân loại dữ liệu trong các hệ thống tự động.

Ngoài ra, thuật toán Naive Bayes cũng tồn tại các ưu điểm và nhược điểm như sau :

- ***Ưu điểm:*** Đơn giản và hiệu quả cho các bài toán phân loại. Hoạt động tốt khi số lượng đặc trưng lớn.
- ***Nhược điểm:*** Giả định về sự độc lập có thể không phù hợp với mọi dữ liệu. Yêu cầu lượng dữ liệu huấn luyện đủ lớn.

Thuật toán Naive Bayes là một công cụ quan trọng và linh hoạt trong lĩnh vực phân loại, đặc biệt là khi áp dụng cho các bài toán văn bản và dự đoán dựa trên xác suất. Sự đơn giản và hiệu quả của nó làm cho nó trở thành một lựa chọn phổ biến trong nhiều ứng dụng thực tế.



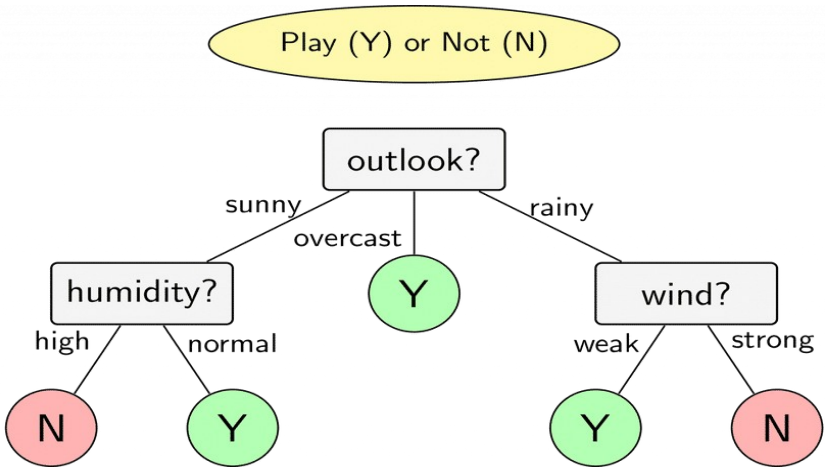
Hình 2.46. Minh họa thuật toán Naive Bayes

❖ *Thuật toán Decision Tree*

Thuật toán Decision Tree (cây quyết định) là một trong những thuật toán quan trọng trong học máy được sử dụng cho cả bài toán phân loại (classification) và hồi quy (regression). Decision Tree tạo ra một cây quyết định dựa trên các quy tắc "nếu... thì..." để dự đoán giá trị đối tượng đầu ra. Thuật toán biểu diễn quyết định dưới dạng cây có nhiều nút và lá. Mỗi nút trong cây đại diện cho một quyết định dựa trên một đặc trưng nào đó, và mỗi lá đại diện cho một giá trị dự đoán.

Decision Tree tìm cách chia dữ liệu dựa trên đặc trưng sao cho thông tin (entropy) hoặc độ thuần khiết (impurity) của mỗi phần được chia là tối đa. Có nhiều thuật toán phổ biến cho việc chia nhánh bao gồm ID3, C4.5, CART. Đối với bài toán phân loại, Decision Tree sử dụng entropy hoặc Gini impurity để đo lường thông tin và độ thuần khiết của mỗi phần được chia. Ngoài ra đối

với bài toán hồi quy, thường sử dụng phương sai (variance) để đo lường độ đồng nhất của dữ liệu.



Hình 2.47. Hình ảnh minh họa thuật toán Decision Tree

Tuy nhiên, mặc dù đơn giản và dễ hiểu nhưng Decision Tree có xu hướng dễ bị overfitting, tức là mô hình quá mức fit với dữ liệu huấn luyện và không tổng quát hóa tốt cho dữ liệu mới. Ngoài ra, đối với các bài toán phân loại phức tạp thì Decision Tree lại tỏ ra không mấy hiệu quả. Trong thử nghiệm lần này của đồ án tốt nghiệp cũng tiến hành thử nghiệm thuật toán Decision Tree với nhiều mức độ sâu khác nhau để đưa ra đánh giá tổng quan nhất.

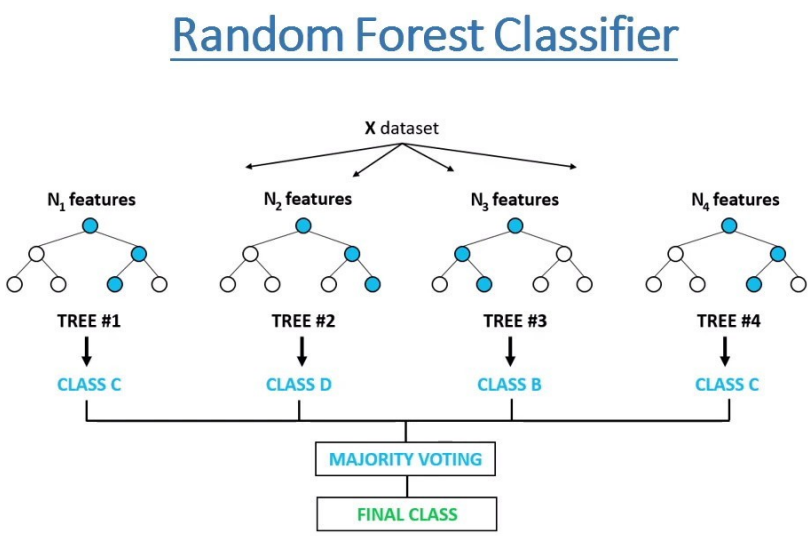
❖ *Thuật toán Random Forest*

Như đã nói bên trên, các thuật toán Decision Tree thường có xu hướng overfitting cho nên cần một phương pháp để hạn chế điều này của thuật toán. Ta có thể sử dụng kỹ thuật giảm cây (pruning) để giúp cho mô hình tránh overfitting hoặc sử dụng thuật toán Random Forest cũng được xây dựng dựa trên thuật toán Decision Tree.

Thuật toán Random Forest sử dụng nhiều cây quyết định (Decision Trees) để tạo ra một "rừng" (forest) cây quyết định, và sau đó kết hợp các dự đoán của chúng để đưa ra dự đoán cuối cùng. Khi xây dựng mỗi cây, Random Forest không sử dụng tất cả các đặc trưng mà chỉ sử dụng một số ngẫu nhiên trong số chúng. Điều này giúp mô hình giảm hiện tượng overfitting và làm cho các cây con trở nên đa dạng.

Mỗi cây con trong Random Forest được xây dựng bằng cách sử dụng một phần của dữ liệu và một phần ngẫu nhiên của đặc trưng. Quá trình này tạo ra các cây con độc lập và đa dạng. Điều này được thực hiện bằng cách lấy mẫu dữ liệu với thay thế từ tập dữ liệu huấn luyện, cho mỗi cây một cách ngẫu nhiên.

Khi cần đưa ra dự đoán cho một điểm dữ liệu mới, Random Forest sử dụng phương pháp voting (đối với bài toán phân loại) hoặc averaging (đối với bài toán hồi quy) trên các dự đoán của từng cây để quyết định kết quả cuối cùng. Do sự đa dạng và độc lập giữa các cây con, Random Forest thường ít bị overfitting hơn so với một cây quyết định đơn.



Hình 2.48. Hình ảnh minh họa thuật toán Random Forest

Thuật toán Random Forest giúp giảm thiểu overfitting so với Decision Tree nhờ sự đa dạng từ việc xây dựng các cây con. Đây cũng là thuật toán được áp dụng rất nhiều trong các bài toán khác nhau trong đó có cả phân loại hình ảnh. Trong thử nghiệm đối với thuật toán Random Forest, đồ án cũng tiến hành thử nghiệm Random Forest với số lượng cây khác nhau để quan sát các kết quả. Việc thực hiện thuật toán này cũng được tiến hành trên thư viện scikit-learn đối với số lượng cây n lần lượt bằng 3,5,7,9,11.

2.3.2. Thuật toán Fourier

Trong phân tích ảnh và xử lý ảnh, thuật toán Fourier thường được sử dụng để phát hiện và đánh giá mức độ mờ trong ảnh. Phương pháp này liên quan đến biến đổi Fourier, một công cụ toán học mạnh mẽ để phân tích các tín hiệu trong không gian tần số. Dưới đây là cách thuật toán Fourier có thể được áp dụng để phát hiện ảnh mờ:

- ***Biến đổi Fourier 2D:*** Đầu tiên, ảnh được chuyển đổi từ không gian thời gian sang không gian tần số bằng cách sử dụng biến đổi Fourier 2 chiều. Điều này tạo ra một biểu diễn tần số của ảnh, trong đó thông tin về các tần số cao và thấp được hiển thị.
- ***Spectrum Analysis:*** Bằng cách phân tích spectrum tần số của ảnh, có thể xác định được các thành phần tần số chính. Các đường vạch sáng trong biểu đồ tần số thường tương ứng với các tần số đáng chú ý trong ảnh.

- Thresholding: Mức độ mờ thường đi kèm với sự mất mát của các tần số cao trong ảnh. Dựa trên phân tích tần số, bạn có thể áp dụng một ngưỡng để đánh giá mức độ mờ. Các tần số cao nằm dưới ngưỡng này có thể được coi là bị mất mát.
- ***Entropy Analysis:*** Một phương pháp khác là phân tích entropy của ảnh trong miền tần số. Mức độ mờ thường đi kèm với sự phân tán và đồng đều của các tần số. Nếu entropy tăng, có thể cho thấy sự không chắc chắn và mất mát thông tin, có thể là dấu hiệu của mức độ mờ.
- ***Gradient Analysis:*** Bạn cũng có thể xem xét gradient của ảnh trong không gian tần số. Sự mờ thường đi kèm với sự giảm độ chói và các đường gradient dịch chuyển về phía zero.

Thuật toán Fourier trong phát hiện ảnh mờ cung cấp một cách tiếp cận toán học để đánh giá sự phân tán của tần số và mất mát thông tin trong ảnh. Tùy thuộc vào ứng dụng cụ thể, các phương pháp khác nhau có thể được kết hợp để tối ưu hóa kết quả.

2.3.3. Mô hình học sâu MobileNet

Trong phạm vi thử nghiệm các thuật toán và mô hình học sâu cho chức năng phát hiện ảnh mờ, đồ án cũng tập trung thử nghiệm với mô hình học sâu MobileNet. Đây là một mô hình được phát triển để dành riêng cho việc tích hợp vào các thiết bị hạn chế về tài nguyên như các thiết bị di động, cảm biến, … Đối với mô hình MobileNet nói chung có 3 phiên bản bao gồm MobileNetV1, MobileNetV2 và MobileNetV3. Trong phạm vi đồ án này sẽ chỉ giới thiệu về hai phiên bản đầu tiên đó là MobileNetV1 và MobileNetV2 vì đây là phiên bản mô hình được đồ án ứng dụng vào thử nghiệm cho tác vụ phát hiện ảnh mờ.

❖ *MobileNetV1*

Đây là 1 kiến trúc neural network được phát triển bởi nhóm các nhà nghiên cứu của Google.Kiến trúc này mang lại kết quả chính xác cao trong khi vẫn giữ các tham số và phép toán ở mức thấp nhất có thể để có khả năng chạy trên các thiết bị di động. MobileNets là các mô hình nhỏ, độ trễ thấp, công suất thấp được tham số hóa để đáp ứng các hạn chế về tài nguyên của nhiều trường hợp sử dụng khác nhau.

Khác với các mô hình xử lý ảnh thông thường, một trong những đặc điểm quan trọng của MobileNet là việc sử dụng *Depthwise Separable Convolution* thay vì Convolution truyền thống. Điều này giúp giảm số lượng tham số và tăng tốc quá trình tính toán, làm cho mô hình phù hợp với các thiết bị có tài nguyên hạn chế. Ngoài ra, mô hình cũng sử dụng Global Average Pooling ở lớp cuối cùng để giảm kích thước của đầu ra và giảm nguy cơ overfitting.

MobileNets được xây dựng chủ yếu từ các dữ liệu có thể phân tách theo chiều sâu và sau đó được sử dụng trong các mô hình Inception để giảm bớt tính toán trong vài lớp đầu tiên. Cấu trúc MobileNet được xây dựng trên các tích chập có thể phân tách theo chiều sâu (depthwise separable convolution) như đã đề cập trong phần trước, ngoại trừ lớp đầu tiên là tích chập đầy đủ. Kiến trúc cụ thể MobileNet được tác giả của bài báo khoa học giới thiệu như hình dưới đây:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Hình 2.49. Hình ảnh minh họa kiến trúc mô hình MobileNet [13]

Tất cả các lớp trên đều được theo sau bởi một batchnorm và một lớp phi tuyến ReLU ngoại trừ lớp kết nối đầy đủ cuối cùng không có lớp phi tuyến và được đưa vào lớp softmax để phân loại. Tóm lại, trong bài báo tác giả trình bày rằng nếu coi tất cả các lớp tích chập có thể phân tách theo chiều sâu (Depthwise Separable Convolution) như một lớp riêng biệt thì mô hình MobileNetV1 này có tổng cộng 28 lớp.

❖ *MobileNetV2*

Cải tiến hơn so với mô hình ban đầu MobileNetV1 thì MobileNetV2 cải thiện hiệu suất của các mô hình di động trên nhiều tác vụ cũng như trên nhiều kích thước mô hình khác nhau. Nó là một trình trích xuất tính năng rất hiệu quả để phát hiện và phân đoạn đối tượng.

Một trong những đặc điểm chính của MobileNetV2 là sử dụng "inverted residuals" để cải thiện khả năng học của mô hình. Điều này bao gồm việc sử dụng các bottleneck layers với kích thước nhỏ để giảm chiều sâu của mô hình, giúp tăng tốc quá trình đào tạo và triển khai. Kế thừa lại mô hình MobileNetV1, MobileNetV2 cũng tiếp tục sử dụng Depthwise Separable Convolution, nhưng thêm vào đó là các Shortcut Connections (Residual Connections). Điều này giúp giảm hiện tượng Vanishing Gradient và cải thiện quá trình học. Ngoài ra, MobileNetV2 cũng lại sử dụng Global Average Pooling để giảm kích thước của đầu ra, giúp giảm nguy cơ overfitting và sử dụng Width Multiplier và Resolution Multiplier để kiểm soát kích thước và độ chính xác của mô hình. Điều này làm cho MobileNetV2 trở thành lựa chọn linh hoạt cho nhiều tài nguyên tính toán khác nhau.

Cấu trúc chi tiết của mô hình MobileNetV2 được thể hiện như hình dưới đây:

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

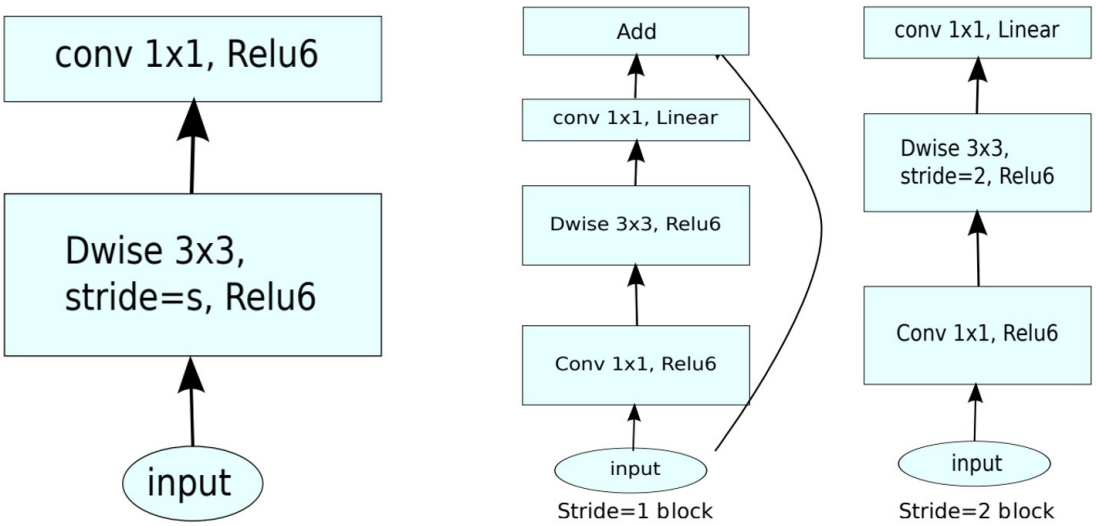
Hình 2.50. Kiến trúc tổng quát mô hình MobileNetV2 [14]

Trong đó, MobileNetV2 chứa đầy đủ 32 bộ lọc, tiếp theo là 19 lớp thắt cổ chai (residual bottleneck layers) như hình 2.13 và sử dụng ReLU6 làm phi tuyến tính.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Hình 2.51. Kiến trúc từng khối mô hình MobileNetV2 [14]

Hình ảnh dưới đây tác giả bài báo cũng chỉ ra them một số khác biệt giữa MobileNatV1 và MobileNatV2. Trong đó, MobileNet gồm sử dụng 1 loại blocks gồm 2 phần, Deepwise và Pointwise, còn MobileNet v2 sử dụng 2 loại blocks, bao gồm: residual block với stride = 1 và block với stride = 2 phục vụ downsizing.



Hình 2.52. Hình ảnh so sánh kiến trúc MobileNetV1 và MobileNetV2 [14]

Có 3 phần đối với mỗi block:

- Layer đầu là 1×1 convolution với ReLU6.

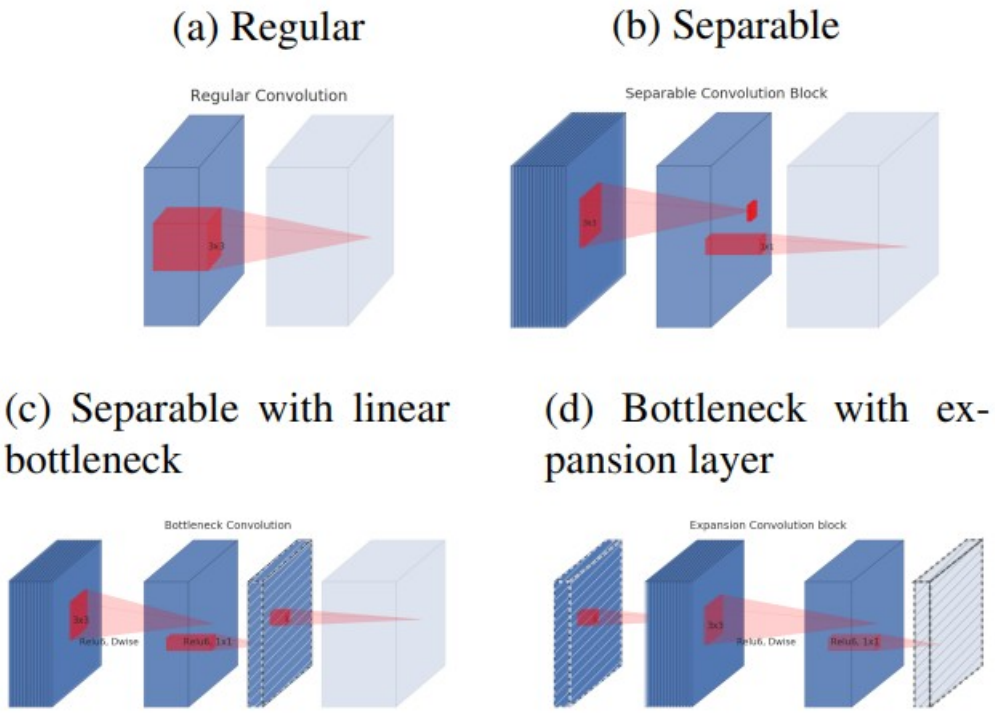
- Layer thứ hai, như cũ, là depthwise convolution.
- Layer thứ 3 tiếp tục là 1×1 convolution nhưng không có activation function. Linear được sử dụng thay vì ReLu như bình thường.

Các kết nối tắt ở MobileNetV2 được điều chỉnh sao cho số input và output channels của mỗi block residual được thắt hẹp lại. Chính vì thế nó được gọi là các bottleneck layers.

Residual block của MobileNetV2 ngược lại so với các kiến trúc residual truyền thống, vì kiến trúc residual truyền thống có số lượng kênh ở input và output của một block lớn hơn so với các layer trung gian. Chính vì vậy nó còn được gọi là Inverted residual block.

Các layer trung gian trong một block sẽ làm nhiệm vụ biến đổi phi tuyến nên cần dày hơn để tạo ra nhiều phép biến đổi hơn. Kết nối tắt giữa các block được thực hiện trên những bottleneck input và output chứ không thực hiện trên các layer trung gian. Do đó các layer bottleneck input và output chỉ cần ghi nhận kết quả và không cần thực hiện biến đổi phi tuyến.

Ở giữa các layer trong một block inverted residual block chúng ta cũng sử dụng những biến đổi tích chập tách biệt chiều sâu để giảm thiểu số lượng tham số của mô hình. Đây cũng chính là bí quyết giúp họ các model MobileNet có kích thước giảm nhẹ.

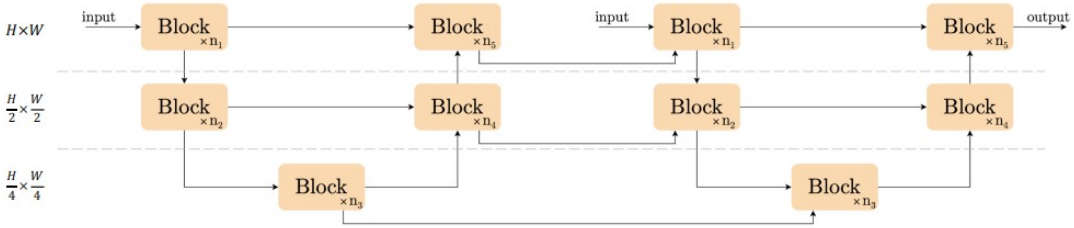


Hình 2.53. So sánh giữa các khối tích chập trong mô hình MobileNet [14]

2.3.4. Mô hình học sâu NAFNet

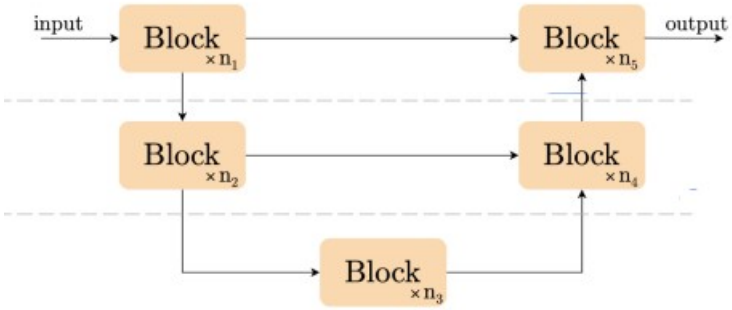
Đối với tác vụ khôi phục ảnh mờ, em thử nghiệm mô hình NAFNet [15] được công bố năm 2022 từ các nhà khoa học Trung Quốc. Điểm nổi bật nhất của kiến trúc mạng này bởi tính đơn giản của nó trong toàn bộ kiến trúc lẫn từng khối kiến trúc nhưng vẫn đạt được kết quả SOTA. Ngoài ra,theo tác giả thì đây là công trình đầu tiên cho thấy rằng các hàm kích hoạt phi tuyến tính có thể được lược bỏ và không cần thiết cho các tác vụ khôi phục ảnh mờ đạt được kết quả SOTA. Kiến trúc mới này đem lại hiệu quả SOTA nhưng lại có độ phức tạp giữa các khối và độ phức tạp trong các khối vô cùng thấp khiến cho việc khôi phục ảnh mờ trở nên nhanh chóng.

Hầu hết các phương pháp dựa trên học sâu đạt được kết quả SOTA đều được xem là các biến thể của mạng U-Net. Nó xếp các khối kiến trúc thành hình chữ U với khả năng bỏ qua các kết nối. Đối với các mô hình khác thì chúng được xây dựng bằng kiến trúc U-Net nhiều giai đoạn, nghĩa là giai đoạn sau tinh chỉnh kết quả cho giai đoạn trước.



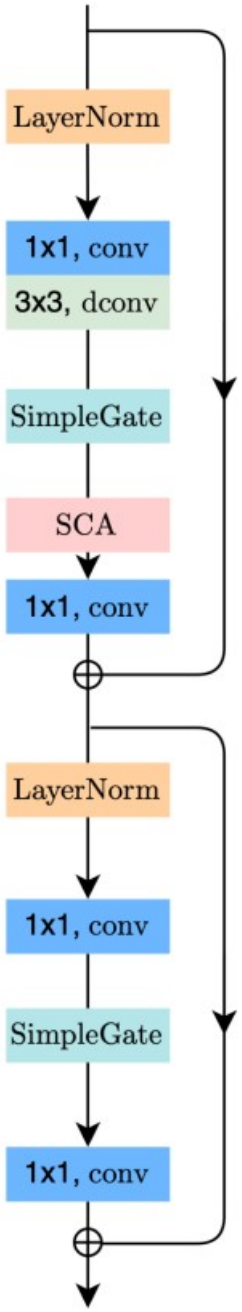
Hình 2.54. Kiến trúc U-Net nhiều giai đoạn của các mạng SOTA [15]

Ngược lại, đối với mạng NAFNet được áp dụng thiết kế mạng U-Net một giai đoạn, nghĩa là chỉ có một tập hợp các khối xếp thành hình chữ U nhưng vẫn đạt được hiệu quả cao.



Hình 2.55. Kiến trúc mạng U-Net một giai đoạn NAFNet [15]

Vừa rồi là cấu trúc tổng thể của mạng NAFNet được giới thiệu ở hình 2.17 Không chỉ đơn giản ở mặt cấu trúc giữa các khối mà mô hình mạng NAFNet còn cố gắng tối giản kiến trúc từng khối đơn để cải thiện khả năng tiêu tốn tài nguyên của mô hình ít hơn so với các mô hình SOTA trước đó. Cụ thể, từng khối trong mạng NAFNet được thiết kế như hình dưới đây:



Hình 2.56. Kiến trúc từng khối của mạng NAFNet [15]

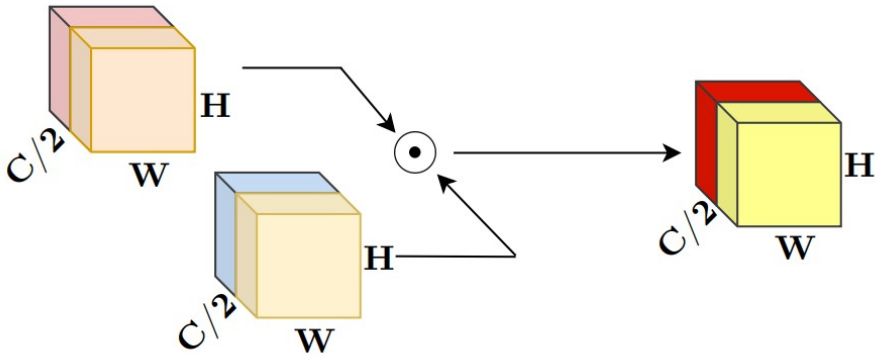
Dưới đây, em sẽ đi phân tích những đặc trưng nổi bật nhất của từng khối trong một block mạng NAFNet.

❖ *Layer Norm*

Layer Norm là lớp chuẩn hóa được áp dụng rộng rãi trong các tác vụ thị giác máy tính cấp cao. Nó được thiết kế để giảm hiện tượng mất thông tin (covariate shift) trong quá trình đào tạo mạng nơ-ron và cải thiện khả năng học của mô hình. Điều này giúp mô hình học được biểu diễn tối ưu hơn bằng cách đảm bảo rằng các tầng ẩn không phải luôn phải đối mặt với các đầu vào có phân phối thay đổi nhiều qua các batch. Điều này có thể hữu ích đặc biệt

khi đối mặt với dữ liệu không ổn định hoặc khi đào tạo mô hình sâu. Ở đây, tác giả quan sát thấy rằng các mô hình trước đây đạt kết quả SOTA đều có sự xuất hiện của Layer Norm và bằng thực nghiệm trong công bố khoa học đó nó đã giúp cải thiện tốc độ học lên gấp 10 lần cũng như đạt được hiệu suất đáng kể trên tập dữ liệu GoPro. Cho nên, tác giả đã them Layer Norm vào đầu mỗi khối vì nó có thể ổn định hóa quá trình đào tạo mô hình

❖ *Simple Gate*



Hình 2.57. Minh họa khối Simple Gate [15]

Simple Gate là kết quả trong quá trình biến đổi toán học của tác giả dựa trên Gated Linear Units được biểu diễn như sau:

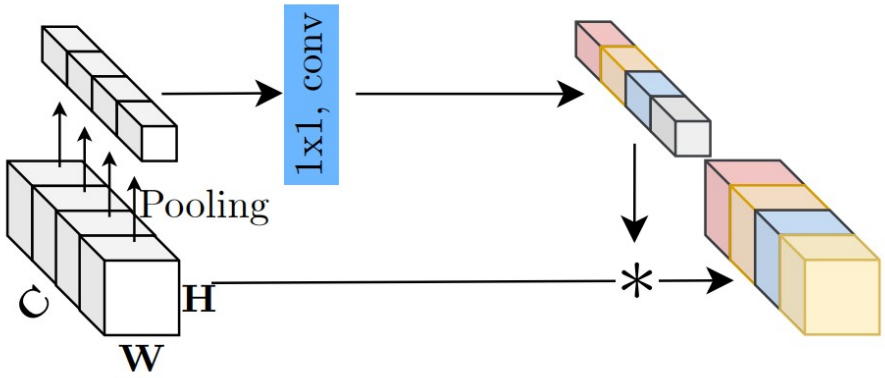
$$Gate(\mathbf{X}, f, g, \sigma) = f(\mathbf{X}) \odot \sigma(g(\mathbf{X})),$$

Trong đó, X đại diện cho các feature map, f và g lần lượt là các biến đổi tuyến tính, σ là một hàm kích hoạt phi tuyến tính và $\odot$ biểu thị cho phép nhân theo phân tử. Từ cách biến đổi toán học kết hợp với công thức hàm kích hoạt GELU [16], tác giả đưa ra kết quả của phép tính Simple Gate như sau:

$$SimpleGate(\mathbf{X}, \mathbf{Y}) = \mathbf{X} \odot \mathbf{Y},$$

Bằng cách sử dụng Simple Gate, tác giả cũng thử nghiệm để chỉ ra mô hình tăng hiệu suất khử nhiễu và khử mờ trên tập dữ liệu GoPro.

❖ *SCA*



Hình 2.58. Minh họa khối SCA [15]

Trong công bố của mình, tác giả cũng đưa ra chứng minh được một khối kích hoạt mới là Simplified Channel Attention (SCA). Tuy không làm cải thiện hiệu suất khử mờ trên tập dữ liệu GoPro nhưng tính đơn giản của khối lại giúp cho mô hình trở nên đơn giản hơn rất nhiều.

Tóm lại, đối với mạng NAFNet tác giả tập trung rút ngắn độ phức tạp từ các mô hình SOTA trước đây nhưng vẫn đạt được hiệu suất tốt. Tác giả chỉ ra các thử nghiệm cho thấy Simple Gate và Simplified Channel Attention (SCA) mặc dù khiến cho mô hình trở nên đơn giản nhưng vẫn đảm bảo hiệu năng của mô hình trong hoạt động khử mờ trên tập dữ liệu thử nghiệm GoPro.

❖ ***Kết luận chương***

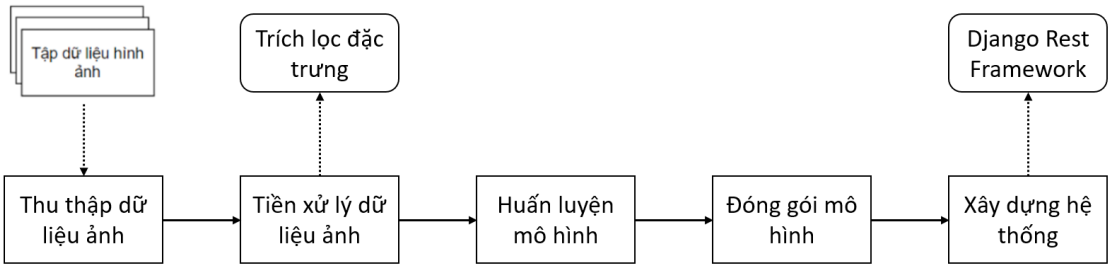
Như vậy xuyên suốt nội dung chương 2 đã thể hiện rõ nét nhất về cách vận hành của hệ thống phần mềm được xây dựng trong phạm vi đồ án. Chương 2 đã làm rõ sơ đồ tổng quát và luồng hoạt động của hai chức năng phát hiện ảnh mờ và khôi phục ảnh mờ. Bên cạnh đó, chương 2 đã trình bày chi tiết các công nghệ được áp dụng vào trong việc xây dựng chương trình phần mềm. Ngoài ra, nội dung lý thuyết về các thuật toán, mô hình học máy, mô hình học sâu đã thể hiện rõ những công việc liên quan đến mô hình mà đồ án sẽ thử nghiệm. Những lý thuyết về thuật toán, mô hình học máy, mô hình học sâu được giới thiệu ở chương 2 cũng là nền tảng để dẫn đến những thử nghiệm sẽ được trình bày ở chương 3. Chương 3, đồ án sẽ làm rõ trình tự thử nghiệm mô hình để đưa ra các nhận xét, đánh giá cụ thể. Ngoài ra, chương 3 cũng sẽ trình bày về các phương thức thực hiện xây dựng chương trình và chạy thử nghiệm để đưa ra kết quả minh họa cuối cùng của chương trình .

CHƯƠNG 3. XÂY DỰNG CHƯƠNG TRÌNH

Nội dung chương 3 sẽ trình bày các bước để xây dựng một mô hình cho hệ thống phần mềm. Bên cạnh đó, nội dung chương 3 sẽ minh họa, thử nghiệm các thuật toán, mô hình được trình bày ở nội dung chương 2. Cuối cùng, chương 3 trình bày các bước xây dựng chương trình và minh họa kết quả chạy chương trình phần mềm. Tuần tự các thành phần của chương 3 bao gồm:

- Giới thiệu chung về các bước xây dựng mô hình
- Tiến hành thử nghiệm các thuật toán, mô hình và đưa ra các kết luận, đánh giá
- Tiến hành xây dựng chương trình phần mềm ở cả Server và Client
- Chạy thử nghiệm chương trình và minh họa các kết quả của từng chức năng

3.1. Giới thiệu chung



Hình 3.59. Minh họa sơ đồ xây dựng hệ thống

Quy trình thực hiện hệ thống phần mềm để phát hiện và khôi phục ảnh mờ được xây dựng trải qua 4 bước chính bao gồm:

- ***Bước 1: Thu thập dữ liệu ảnh***

Đây là bước thu thập các dữ liệu liên quan đến ảnh mờ chuyển động, ảnh mờ do nhiễu từ các bộ dữ liệu có sẵn và các trang mạng xã hội. Đây là công đoạn quan trọng và đòi hỏi sự kiên nhẫn khi phải chắc lọc các hình ảnh phù hợp với bài toán của đề tài đồ án.

- ***Bước 2: Xử lý ảnh và trích chọn đặc trưng***

Tiền xử lý ảnh là một công đoạn nhằm để khai thác tối đa dữ liệu ảnh bằng cách chuyển đổi từ dữ liệu ảnh thô thu thập được ở bước 1 sang các đặc trưng nổi bật phù hợp với bài toán của đề tài. Tại bước này bao gồm các quá trình như tổng hợp dữ liệu, phân loại dữ liệu và trích chọn đặc trưng ảnh.

- ***Bước 3: Thử nghiệm và đánh giá các mô hình***

Sau khi có đầy đủ các dữ liệu, các thuộc tính cần thiết thì đây sẽ là bước khởi tạo và thử nghiệm huấn luyện các mô hình học máy, mô hình học sâu đã nêu ở mục 2.3. Từ đó đưa ra các đánh giá khách quan với các loại mô hình khác nhau cho bài toán của đề tài

- ***Bước 4: Chọn mô hình phù hợp và xây dựng hệ thống***

Cuối cùng, dựa vào các đánh giá từ các mô hình, thuật toán thử nghiệm huấn luyện để chọn ra một thuật toán, mô hình phù hợp cho ứng dụng của đề tài. Sản phẩm ứng dụng chính là kết quả cuối cùng đúc kết lại toàn bộ quá trình nghiên cứu và thử nghiệm các mô hình của đề tài đồ án này.

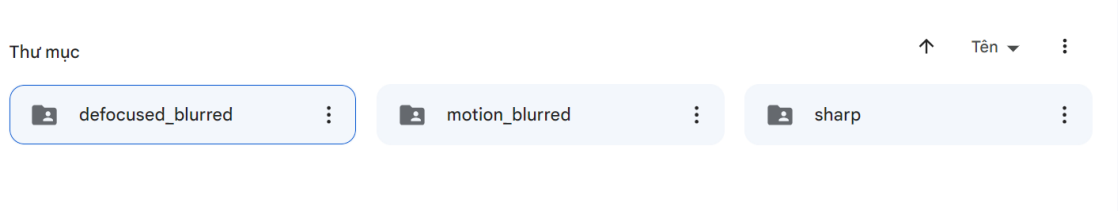
3.2. Các bước thực hiện

3.2.1. Thu thập dữ liệu ảnh

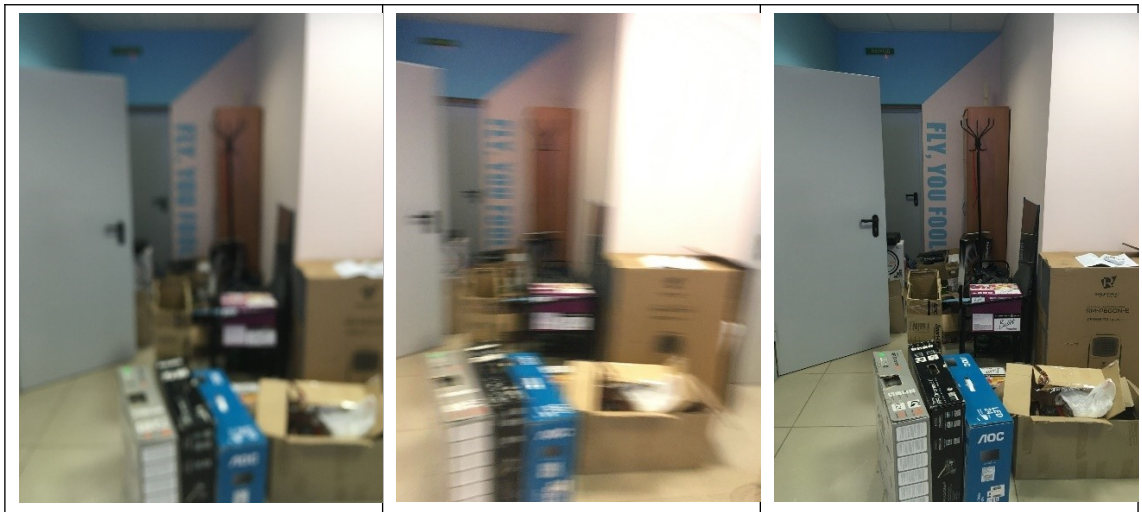
Tập dữ liệu ảnh cho hai tác vụ phát hiện ảnh mờ và khôi phục ảnh mờ được sưu tập và tổng hợp như sau:

❖ *Tập dữ liệu cho phát hiện ảnh mờ*

Tập dữ liệu thử nghiệm bao gồm 1050 ảnh bao gồm 350 bộ ba ảnh cùng một cảnh với ba phân loại sắc nét (sharp), nhòe mờ (defocused blur) và ảnh chuyển động mờ (motion blur) như ví dụ dưới đây:



Hình 3.60. Các folder dữ liệu tương ứng với ba loại ảnh



Defocus blur image	Motion blur image	Sharp image

Hình 3.61. Minh họa ba loại ảnh đối với một khung cảnh

Đặc điểm của bộ dữ liệu:

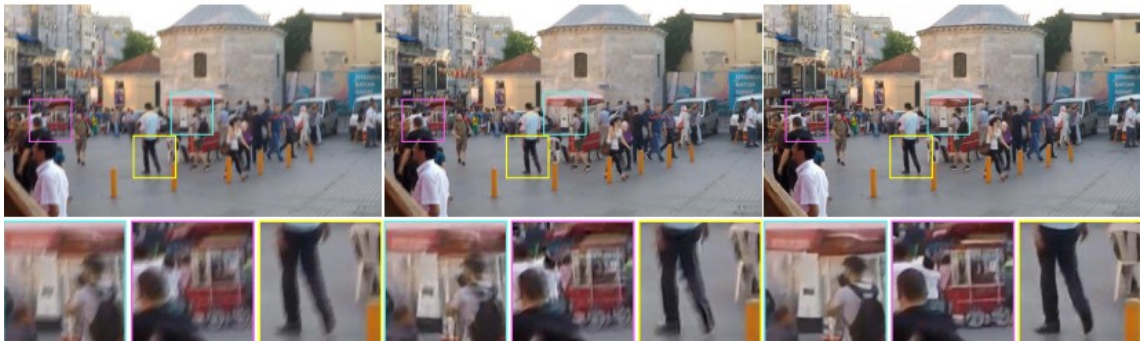
- Bộ dữ liệu bao gồm các hình ảnh có định dạng: JPG
- Bộ dữ liệu ảnh trên được chụp từ 66 thiết bị khác nhau, chủ yếu là điện thoại thông minh nhưng cũng có một số máy ảnh chuyên dụng.
- Các hình ảnh được sưu tầm, chắt lọc trên các bộ dữ liệu có sẵn và tài nguyên trên các trang mạng xã hội.

- Các hình ảnh này đều được phân loại và lọc rõ vào từng folder phù hợp với nó để phục vụ cho quá trình huấn luyện mô hình sau này.

❖ *Tập dữ liệu cho khôi phục ảnh mờ*

Tập dữ liệu được sử dụng cho khôi phục ảnh mờ là tập dữ liệu có sẵn GoPro.

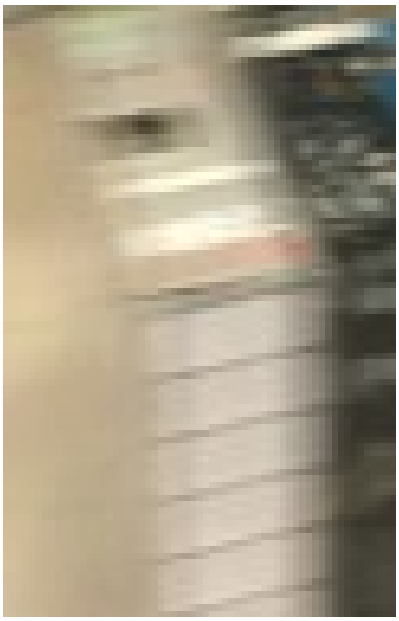
- Đây là tập dữ liệu bao gồm 3214 ảnh mờ với kích thước 1280 x 720.
- Tập dữ liệu này được chia làm 2103 ảnh cho tập dữ liệu huấn luyện (train set) và 1111 ảnh cho tập kiểm thử (test set).
- Đây là bộ dữ liệu bao gồm các cặp hình ảnh mờ thực tế và hình ảnh sắc nét thực tế mặt đất tương ứng thu được bằng camera tốc độ cao.
- Tập dữ liệu GoPro này được tác giả đăng tải và chia sẻ để giới thiệu tới cộng đồng tại địa chỉ https://seungjunnah.github.io/Datasets/gopro



Hình 3.62. Hình ảnh minh họa tập dữ liệu GoPro

3.2.2. Xử lý ảnh và trích chọn đặc trưng

Dựa trên một quan sát cơ bản ta có thể thấy rằng, những hình ảnh bị mờ thường có xu hướng không thể nhận diện các cạnh của ảnh. Một hình ảnh nét thì các cạnh của ảnh sẽ có sự nhận biết rõ ràng hơn một tấm ảnh mờ. Trong một tấm ảnh mờ thường có xu hướng gần màu nhau đối với các pixel liền kề nhau. Cụ thể, nếu ta nhìn một tấm ảnh bị mờ do nhòe hay do chuyển động, ta có thể thấy rằng các sẽ có các vùng lân cận vật thể chứa các pixel gần màu với vật thể. Như hình minh họa 3.5 dưới đây, với vật thể là một chiếc hộp màu xám, khi tấm ảnh bị nhòe đi dẫn đến các pixel xung quanh chiếc hộp cũng có màu tương tự màu xám của chiếc hộp, điều này ta hoàn toàn có thể nhìn thấy bằng mắt thường.



Hình 3.63. Minh họa ảnh chụp vật thể bị mờ do rung lắc

Chính vì lý do trên, đồ án sẽ tiến hành việc trích chọn đặc trưng của ảnh bằng cách thử nghiệm các thuật toán phát hiện cạnh được giới thiệu dưới đây.

❖ *Toán tử Sobel*

Toán tử Sobel là một phương pháp thường được sử dụng trong xử lý ảnh để phát hiện cạnh (edge detection). Được đặt tên theo tên của nhà toán học và kỹ sư người Mỹ Irwin Sobel, toán tử này nhằm xác định sự thay đổi độ chói giữa các điểm ảnh xung quanh trong một hình ảnh.

Toán tử Sobel thường được sử dụng để xác định đạo hàm của hàm mức xám của ảnh, giúp tìm ra các vùng cạnh và đồng thời làm nổi bật chúng. Cụ thể, toán tử Sobel được thực hiện bằng cách sử dụng hai ma trận convolution (kernels) để tính toán đạo hàm theo chiều ngang và chiều dọc.

Công thức của toán tử Sobel theo chiều ngang (Gx) và chiều dọc (Gy) là:

- Đối với chiều ngang công thức sử dụng kernel (Gx) nhân với ảnh gốc (I) như sau:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

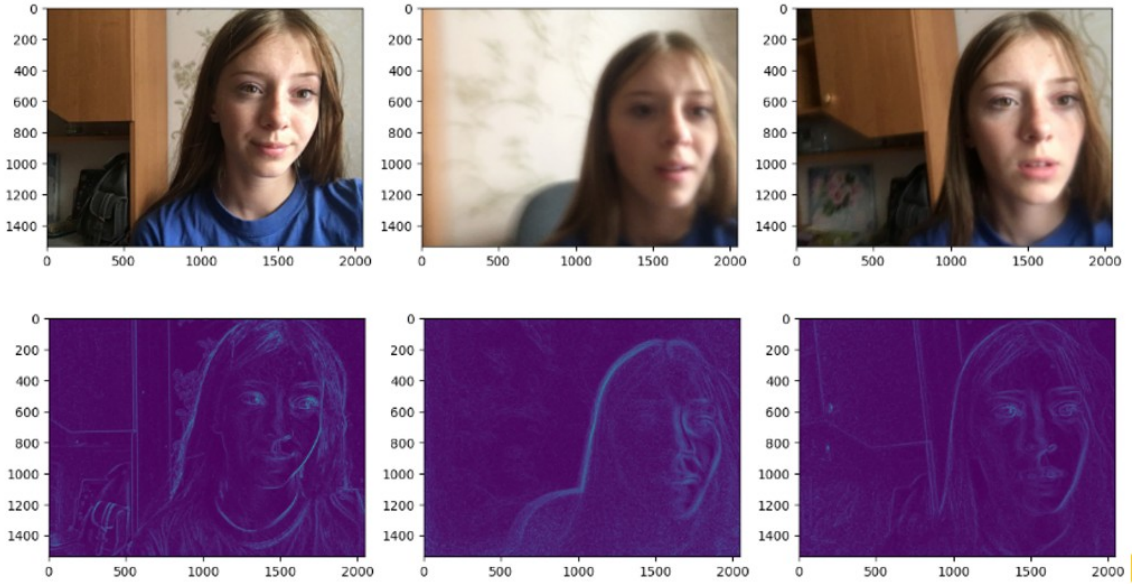
- Đối với chiều dọc, công thức sử dụng kernel (Gy) nhân với ảnh gốc (I) như sau:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

Toán tử Sobel sử dụng các kernel này để tính đạo hàm riêng theo chiều ngang và chiều dọc. Sau đó, độ lớn của đạo hàm có thể được tính bằng công thức:

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

Hình ảnh kết quả của toán tử Sobel thường là một hình ảnh mới, nơi các điểm cạnh có độ lớn lớn sẽ được làm nổi bật và các vùng không có cạnh sẽ có giá trị thấp hơn.



Hình 3.64. Hình ảnh minh họa áp dụng toán tử phát hiện biên

❖ *Toán tử Laplace*

Toán tử Laplace, còn được gọi là toán tử Laplacian, là một công cụ quan trọng trong xử lý ảnh và xử lý tín hiệu. Toán tử này được sử dụng để phát hiện các vùng không gian của một hình ảnh hay tín hiệu mà có sự thay đổi nhanh chóng về độ chói hoặc giảm giá trị tại các điểm ảnh. Nó thường được sử dụng để tìm cạnh, điểm cực trị (cực đại hoặc cực tiểu), và các vùng thay đổi nhanh trong ảnh.

Toán tử Laplace được biểu diễn bằng gradient của gradient, có thể được tính bằng cách lấy đạo hàm riêng cấp hai của hàm mức xám. Đối với hàm hai chiều $I(x, y)$, toán tử Laplace có thể được biểu diễn như sau:

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Một biến thể phổ biến của toán tử Laplace trong xử lý ảnh là sử dụng kernel convolution sau:

$$\nabla^2 I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Toán tử Laplace thường tạo ra các giá trị lớn ở các vùng có sự thay đổi đột ngột trong độ chói, ví dụ như cạnh của đối tượng trong ảnh. Trong môi trường xử lý tín hiệu, toán tử Laplace cũng được sử dụng để đánh giá độ biến động của một tín hiệu trong thời gian. Chính vì công dụng tìm cạnh nhanh chóng của toán tử này, em cũng tập trung vào áp dụng để trích xuất các các đặc trưng về cạnh của tấm ảnh.

Như vậy, đối với hai thuật toán phát hiện cạnh được giới thiệu trên và dựa trên tính chất đã giới thiệu về ảnh mờ nên em đã áp dụng để trích lọc ra 3 đặc trưng từ một tấm ảnh bao gồm:

- Phương sai (variance): tổng bình phương khoảng cách từng phần tử đối với phần tử trung bình. Thật vậy, có thể dễ dàng suy luận rằng phương sai ảnh biên nét sẽ lớn hơn phương sai ảnh biên mờ.
- Trung bình (mean): Trung bình cộng tất cả phần tử, đối với ảnh nét sẽ lớn hơn
- Giá trị lớn nhất (max): giá trị lớn nhất ghi nhận được ở 1 điểm biên, nếu đạt 255 hoặc gần 255 thì khả năng cao là ảnh nét.

Sau khi trích chọn được các đặc trưng cơ bản trên và dựa vào bộ dữ liệu thử nghiệm đã sưu tầm sẵn, sẽ trích rút ra 3 đặc trưng đã nêu và lưu vào một file csv như đoạn mã dưới đây:

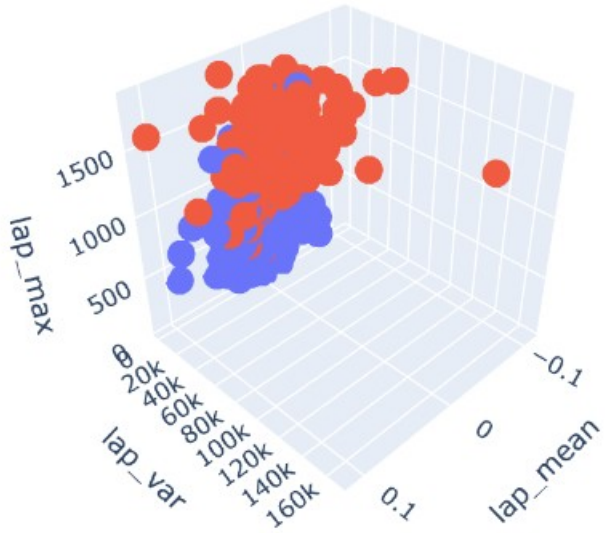
```
general_path = '/content/gdrive/MyDrive/BlurryImages'
data = pd.read_csv(f'{general_path}/data.csv')
# data = data.iloc[0:, 1:]
print(data.shape)
data.head()
```

```
(1050, 5)
```

	image_name	lap_mean	lap_var	lap_max	label
0	0_IPHONE-SE_F.JPG	-0.002243	188.278620	224	blurry
1	100_NIKON-D3400-35MM_F.JPG	-0.000587	484.732559	528	blurry
2	101_NIKON-D3400-35MM_F.JPG	-0.000473	243.201813	244	blurry
3	102_NIKON-D3400-35MM_F.JPG	-0.007294	348.402718	470	blurry
4	103_HUAWEI-P20_F.jpg	-0.013786	178.553902	192	blurry

Hình 3.65. Minh họa đoạn mã trích lọc đặc trưng và lưu vào tập dữ liệu mới

Như vậy, sau khi trích lọc được 3 đặc trưng cơ bản đã nêu, tiến hành quan sát xem các điểm dữ liệu được phân bố trên một không gian 3 chiều như sau:



Hình 3.66. Minh họa sự phân bố của các điểm dữ liệu trong không gian 3 chiều

Dựa vào hình ảnh trên, có thể thấy rằng với 3 đặc trưng đã nếu thì tập dữ liệu cũng đem lại tính phân cụm khá cao, trong đó các điểm dữ liệu màu đỏ đại diện cho các đặc trưng của tấm ảnh nét và điểm dữ liệu màu xanh thì đại diện cho các đặc trưng của tấm ảnh mờ. Điều này cũng là tiền đề để thử nghiệm các thuật toán học máy cơ bản đối với tập dữ liệu sau xử lý này.

3.2.3. Thử nghiệm và đánh giá các mô hình

Thử nghiệm và đánh giá mô hình là công đoạn cuối cùng trong việc xây dựng mô hình để dự đoán và trả về kết quả cuối cùng cho tác vụ phát hiện và khôi phục ảnh mờ. Tiến hành thử nghiệm nhiều mô hình sẽ giúp cho việc đánh giá các mô hình trong một tác vụ của đề tài trở nên khách quan hơn. Sau đây, sẽ là phần nội dung về việc huấn luyện và đánh giá các mô hình, thuật toán đã được giới thiệu ở mục 2.3:

❖ *Thử nghiệm các mô hình học máy Machine Learning*

Ở mục 2.3, đồ án đã giới thiệu các mô hình học máy truyền thống Machine Learning để áp dụng cho tác vụ phát hiện ảnh mờ. Các thuật toán được thực hiện dựa trên bộ dữ liệu sau khi trích rút 3 đặc trưng từ mục 3.2.2. Sau khi đưa tập dữ liệu vào các thuật toán học máy cơ bản đã nêu được thực hiện bởi thư viện Scikit-learn của Python và đạt được các kết quả được tổng hợp như hình dưới đây:

Mô hình	Accuray		Mô hình	Accuracy
Knn k = 12	0.9111		Random forest n = 3	0.8825
Knn k = 13	0.9048		Random forest n = 5	0.8921
Knn k = 14	0.9079		Random forest n = 7	0.9016
Knn k = 15	0.9016		Random forest n = 9	0.9079
Knn k = 16	0.9048		Random forest n = 11	0.8984
Knn k = 17	0.9111		Decision tree maxDepth=1	**0.9175**
Knn k = 18	0.9079		Decision tree maxDepth=2	0.9175
Naive Bayes	0.8889		Decision tree maxDepth=3	0.8984

Bảng 3-2. Kết quả thực nghiệm các mô hình học máy truyền thống

Từ các kết quả trên có thể thấy rằng việc sử dụng các thuật toán học máy dựa trên ba đặc trưng được trích lọc cũng đem lại kết quả dự đoán chính xác khá cao. Độ chính xác cao nhất thuộc về thuật toán cây quyết định với chiều sâu là 1 đem lại độ chính xác lên tới 91,75%. Các kết quả trên cũng đem lại cở sở để hướng tới các thử nghiệm với các thuật toán khác để cải thiện độ chính xác cho bài toán phân loại ảnh mờ này.

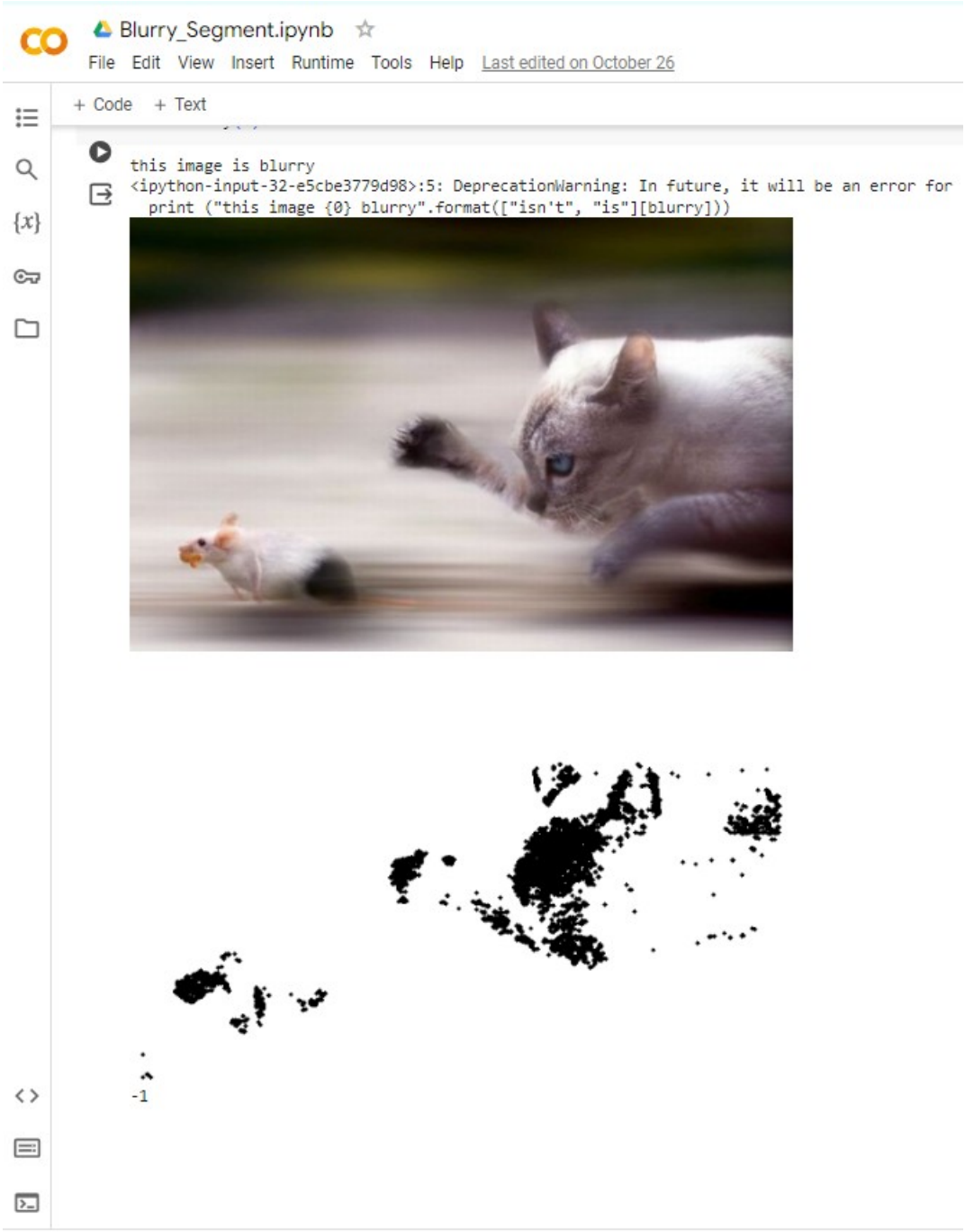
❖ *Thử nghiệm thuật toán Fourier*

Các bước thực hiện biến đổi Fourier để dự đoán khả năng mờ của tấm ảnh như sau:

- Ta chuyển đổi tấm ảnh màu RGB sang ảnh xám giúp tiết kiệm bộ nhớ và tăng tốc độ xử lý thuật toán.
- Chọn ra tọa độ tâm của bức ảnh.
- Thực hiện biến đổi Fourier 2D trên ảnh xám sẽ cho ra một kết quả là một ma trận phức gồm các phần tử biểu diễn biên độ và pha của tần số tại mỗi điểm trên ảnh.
- Di chuyển tần số thấp (gần tâm) của biến đổi Fourier đến tâm của ma trận kết quả. Điều này làm cho tần số thấp được căn giữa, và tần số cao (nằm ở biên) được di chuyển ra xa tâm
- Trong biến đổi Fourier, các tần số cao thường đại diện cho chi tiết nhỏ và nhiễu trong ảnh cho nên ta loại bỏ một phần của tần số cao bằng cách đặt giá trị của chúng là 0.
- Thực hiện tiếp biến đổi Fourier nghịch đảo để chuyển từ không gian tần số trở lại không gian ảnh. Kết quả là một ma trận phức.
- Lấy trung bình các pixel của tấm ảnh để xét với một ngưỡng cho trước xác định độ mờ của tấm ảnh

Các bước thực hiện để phân vùng mờ dựa trên biến đổi Fourier:

- Bước đầu thực hiện biến đổi Fourier trên tấm ảnh như giai đoạn dự đoán độ mờ của tấm ảnh
- Kết quả đầu ra của bước trên được xử lý tiếp bằng cách loại bỏ đi các vùng biên không cần thiết của tấm ảnh để tạo sự tập trung vào các điểm mờ của bức ảnh
- Áp dụng các toán tử **Erosion** và **Dilation** để loại bỏ các nhiễu, loại bỏ các biên và phát hiện, cách biệt các đối tượng.



Hình 3.67. Kết quả thực nghiệm với thuật toán Fourier

Như vậy, dựa trên nền tảng thuật toán biến đổi Fourier kết hợp với một vài kĩ thuật xử lý ảnh khác không chỉ đánh giá khả năng mờ của một tấm ảnh

mà còn có thể phân vùng mờ cho tấm ảnh dựa trên một ngưỡng nhất định. Đối với thực nghiệm này các điểm ảnh được biến đổi so sánh với ngưỡng bằng 1. Các điểm ảnh sau khi biến đổi Fourier nằm trong khoảng 0 và 1 sẽ được coi là những điểm ảnh mờ và được chuyển đổi thành pixel trắng. Ngược lại, các điểm ảnh ngoài ngưỡng 0-1 sẽ được coi là điểm ảnh nét và chuyển đổi thành pixel đen. Từ đây thuật toán sẽ phân vùng được các chi tiết điểm ảnh mờ và nét như kết quả hình 3.10. Có thể thấy rõ bằng mắt thường, các chi tiết bên trong chú mèo và con chuột là các chi tiết nét và các chi tiết bên ngoài dường như đã bị mờ đi do chuyển động.

❖ *Thử nghiệm mô hình học sâu MobileNet*

Trong phạm vi thử nghiệm mô hình, dữ liệu bao gồm 700 ảnh mờ và 350 ảnh nét được chia làm tập huấn luyện (train set) và tập kiểm thử (test set). Trong đó, đối với tập dữ liệu huấn luyện dành cho ảnh mờ bao gồm 500 ảnh được chọn ngẫu nhiên và 200 ảnh cho tập dữ liệu kiểm thử.

```
path = "/content/drive/MyDrive/SRV/detect_dataset"
listImage = [] #đường dẫn những ảnh mờ

for i in os.listdir(path):

    if i=="sharp" or i=="blur_dataset_scaled":

        continue

    for j in os.listdir(path+"/"+i):


        try:

            if j.split(".")[1]!="JPG" and j.split(".")[1]!="jpg" and j.split(".")[1]!="jpeg":

                pass
        except:

            pass

        listImage.append(path+"/"+i+"/"+j)


random.shuffle(listImage) # trộn các đường dẫn ảnh mờ lên
train_blur = listImage[0:500] # lấy 500 ảnh đầu làm train data
test_blur = listImage[500:] # lấy các ảnh từ ảnh 600 đến hết làm test data
print(len(listImage))
print(len(train_blur))
print(len(test_blur))
```

```
700
500
200
```

Hình 3.68. Đoạn mã chia dữ liệu huấn luyện đối với tập dữ liệu ảnh mờ

Ngoài ra, đối với tập dữ liệu ảnh nét được chia thành 300 ảnh cho tập dữ liệu huấn luyện và 50 ảnh cho tập dữ liệu kiểm thử.

```
path = "/content/drive/MyDrive/SRV/detect_dataset"
listImage = [] # đường dẫn những ảnh nét

for i in os.listdir(path):

    if i!="sharp":

        continue

    for j in os.listdir(path+"/"+i):


        try:

            if j.split(".")[1]!="JPG" and j.split(".")[1]!="jpg" and j.split(".")[1]!="jpeg":

                pass
        except:

            pass

        listImage.append(path+"/"+i+"/"+j)


random.shuffle(listImage)
train_sharp = listImage[0:300]
test_sharp = listImage[300:]
print(len(train_sharp))
print(len(test_sharp))
```

```
300
50
```

Hình 3.69. Đoạn mã chia dữ liệu huấn luyện với tập dữ liệu ảnh nét

Đầu vào mô hình bao gồm các ma trận pixel ảnh đã được chuẩn hóa, kích thước ảnh 900x900 và chia thành các batch size = 16.

```
[ ]  image_batch, label_batch = next(val_generator)
     image_batch.shape, label_batch.shape
```

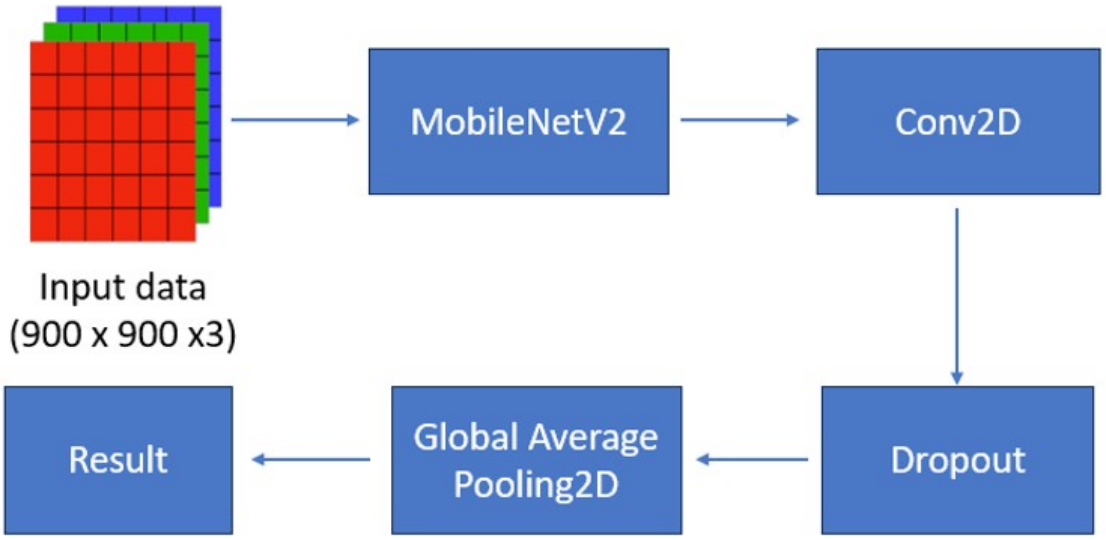
```
((16, 900, 900, 3), (16, 2))
```

Hình 3.70. Đoạn mã chia batch size cho quá trình huấn luyện

Kiến trúc mô hình đơn giản với lớp đầu tiên là mô hình MobileNetV2, tiếp đó là lần lượt các lớp tích chập Conv2D, Dropout và GlobalAveragePooling2D. Cuối cùng là layer cuối với 2 node đầu ra cho hai nhãn ảnh mờ và ảnh nét để phân loại bằng hàm kích hoạt softmax.

```
[ ]  model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, 29, 29, 1280)      2257984
 tional)

 conv2d (Conv2D)             (None, 27, 27, 32)        368672

 dropout (Dropout)           (None, 27, 27, 32)        0

 global_average_pooling2d (  (None, 32)                0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 2626722 (10.02 MB)
Trainable params: 368738 (1.41 MB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Hình 3.71. Tổng quát kiến trúc mô hình huấn luyện



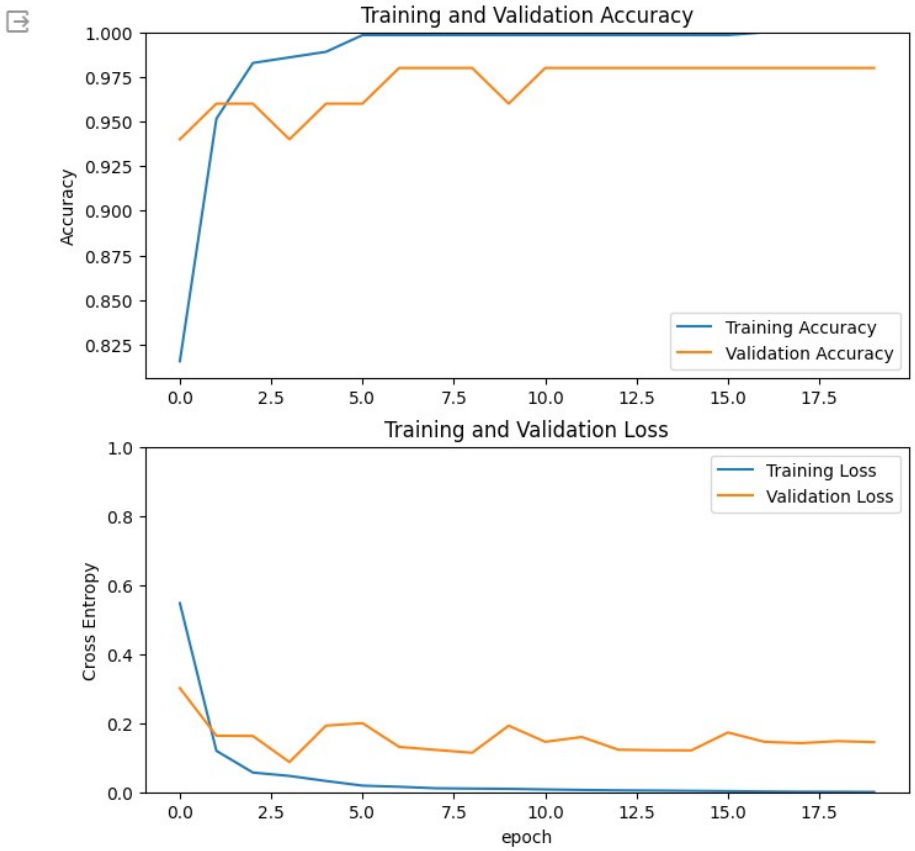
Hình 3.72. Sơ đồ huấn luyện

Trong thử nghiệm lần này, em thực hiện huấn luyện mô hình với 20 epochs và đạt được kết quả đánh giá trên tập dữ liệu validation lên tới độ chính xác (accuracy) là 98%. Kết quả này cho thấy mô hình của em có vẻ đang hoạt động khá tốt với dữ liệu được sưu tầm. Quá trình huấn luyện được em thực hiện trên Google Colaboratory và thể hiện ở hình bên dưới đây:

```
                    validation_steps=len(val_generator))
Epoch 1/20
40/40 [==============================] - 52s 952ms/step - loss: 0.5472 - accuracy: 0.8156 - val_loss: 0.3013 - val_accuracy: 0.9400
Epoch 2/20
40/40 [==============================] - 36s 886ms/step - loss: 0.1197 - accuracy: 0.9516 - val_loss: 0.1634 - val_accuracy: 0.9600
Epoch 3/20
40/40 [==============================] - 36s 880ms/step - loss: 0.0568 - accuracy: 0.9828 - val_loss: 0.1627 - val_accuracy: 0.9600
Epoch 4/20
40/40 [==============================] - 40s 1s/step - loss: 0.0470 - accuracy: 0.9859 - val_loss: 0.0871 - val_accuracy: 0.9400
Epoch 5/20
40/40 [==============================] - 37s 930ms/step - loss: 0.0324 - accuracy: 0.9891 - val_loss: 0.1927 - val_accuracy: 0.9600
Epoch 6/20
40/40 [==============================] - 35s 865ms/step - loss: 0.0190 - accuracy: 0.9984 - val_loss: 0.1998 - val_accuracy: 0.9600
Epoch 7/20
40/40 [==============================] - 35s 874ms/step - loss: 0.0157 - accuracy: 0.9984 - val_loss: 0.1312 - val_accuracy: 0.9800
Epoch 8/20
40/40 [==============================] - 35s 877ms/step - loss: 0.0114 - accuracy: 0.9984 - val_loss: 0.1225 - val_accuracy: 0.9800
Epoch 9/20
40/40 [==============================] - 35s 865ms/step - loss: 0.0104 - accuracy: 0.9984 - val_loss: 0.1142 - val_accuracy: 0.9800
Epoch 10/20
40/40 [==============================] - 34s 856ms/step - loss: 0.0096 - accuracy: 0.9984 - val_loss: 0.1925 - val_accuracy: 0.9600
Epoch 11/20
40/40 [==============================] - 36s 892ms/step - loss: 0.0079 - accuracy: 0.9984 - val_loss: 0.1458 - val_accuracy: 0.9800
Epoch 12/20
40/40 [==============================] - 34s 844ms/step - loss: 0.0064 - accuracy: 0.9984 - val_loss: 0.1597 - val_accuracy: 0.9800
Epoch 13/20
40/40 [==============================] - 34s 844ms/step - loss: 0.0053 - accuracy: 0.9984 - val_loss: 0.1230 - val_accuracy: 0.9800
Epoch 14/20
40/40 [==============================] - 35s 866ms/step - loss: 0.0047 - accuracy: 0.9984 - val_loss: 0.1216 - val_accuracy: 0.9800
Epoch 15/20
40/40 [==============================] - 34s 853ms/step - loss: 0.0038 - accuracy: 0.9984 - val_loss: 0.1210 - val_accuracy: 0.9800
Epoch 16/20
40/40 [==============================] - 35s 876ms/step - loss: 0.0030 - accuracy: 0.9984 - val_loss: 0.1729 - val_accuracy: 0.9800
Epoch 17/20
40/40 [==============================] - 35s 866ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.1458 - val_accuracy: 0.9800
Epoch 18/20
40/40 [==============================] - 34s 849ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.1420 - val_accuracy: 0.9800
Epoch 19/20
40/40 [==============================] - 36s 883ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.1478 - val_accuracy: 0.9800
Epoch 20/20
40/40 [==============================] - 35s 863ms/step - loss: 6.5942e-04 - accuracy: 1.0000 - val_loss: 0.1449 - val_accuracy: 0.9800
```

Hình 3.73. Minh họa quá trình huấn luyện mô hình MobileNet

Sau khi thực hiện việc huấn luyện thành công, em sinh ra đồ thị của các tham số liên quan đến độ chính xác (accuracy) và độ mất mát (loss) của tập train và valid như sau:



Hình 3.74. Biểu đồ minh họa accuracy và loss trên tập training và validation

Từ hình ảnh minh họa trên có thể thấy rằng trong quá trình huấn luyện, mô hình có xu hướng học hỏi tốt dần lên với các tham số loss giảm dần và độ chính xác dự đoán tăng dần. Sau 10 epochs đầu, mô hình đã dần dần đưa ra được các kết quả tích cực chứng tỏ mô hình MobileNet hoạt động tốt với tập dữ liệu ảnh này.

❖ *Thử nghiệm mô hình học sâu NAFNet*

Trong thử nghiệm của mình, em huấn luyện mô hình trên Google Colaboratory trên tập dữ liệu trích xuất từ dữ liệu có sẵn GoPro với 701 ảnh.

Các tham số khởi tạo mô hình

- optimizer: AdamW – đây là một biến thể của thuật toán tối ưu Adam, được thiết kế để giải quyết một số vấn đề của Adam liên quan đến regularization và hạn chế mức tăng thêm trọng số quá mức trong quá trình đào tạo mô hình.
- learning rate: 0.001
- weight_decay: 0.001
- betas: [0.9 , 0.9] – đây là hai chỉ số beta lần lượt giúp kiểm soát mức độ cập nhật trung bình động của gradient và kiểm soát mức độ cập nhật của bình phương gradient.
- loss function: PSNRLoss – đây là hàm mất mát được sử dụng trong quá trình đào tạo công tác xử lý ảnh để đo lường sự khác biệt giữa hình ảnh dự đoán và hình ảnh thực tế.
- reduction: mean – đây là giá trị mất mát của từng điểm dữ liệu được tính và sau đó được lấy giá trị trung bình. Điều này giúp giảm giá trị mất mát xuống một giá trị số duy nhất, tức là trung bình của mất mát trên toàn bộ tập dữ liệu.
- metric: PSNR và SSIM – đây là hai chỉ số đã được giới thiệu ở nội dung 1.7.6 và 1.7.7 để so sánh chất lượng ảnh thực tế với ảnh được sinh ra.

Các thông số huấn luyện mạng

- Tập dữ liệu train bao gồm 701 ảnh và tập test bao gồm 370 ảnh
- Epochs bao gồm 15 epochs với số iter là 20000
- Batch_size: 4

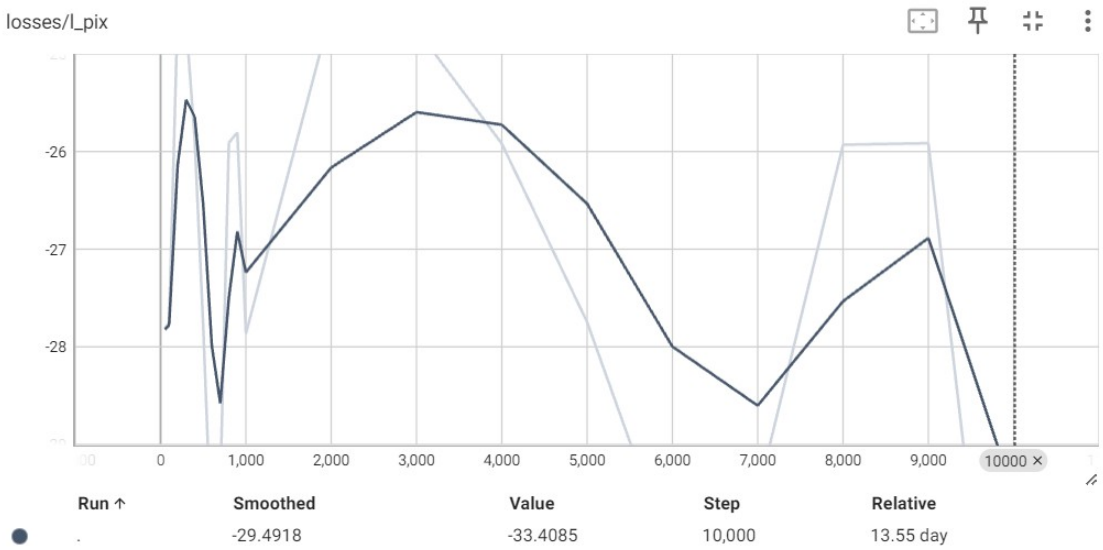
Môi trường huấn luyện

- Google Colaboratory
- GPT T4

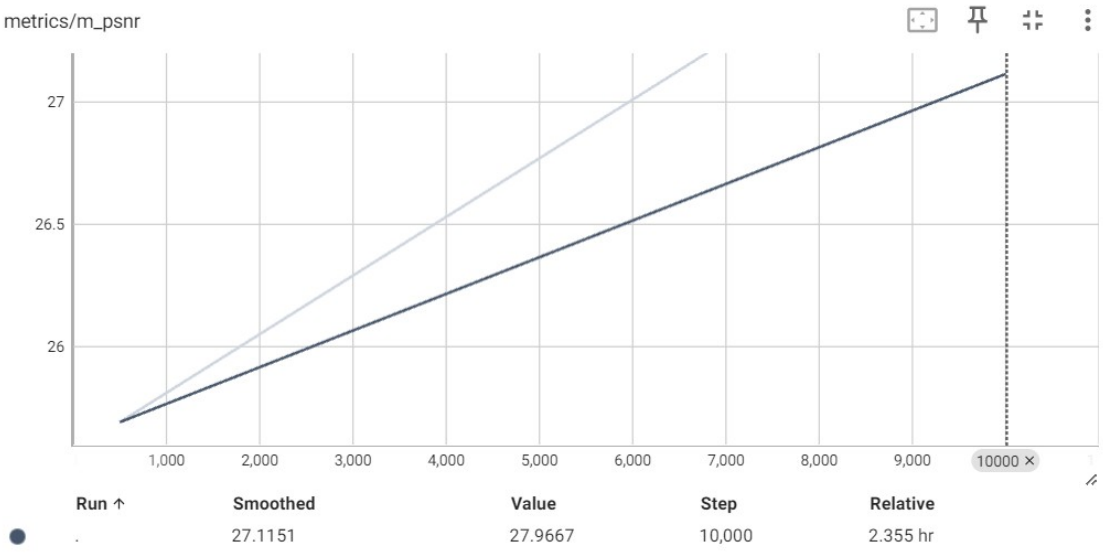
Đánh giá mô hình

- Tổng thời gian huấn luyện: 6 giờ

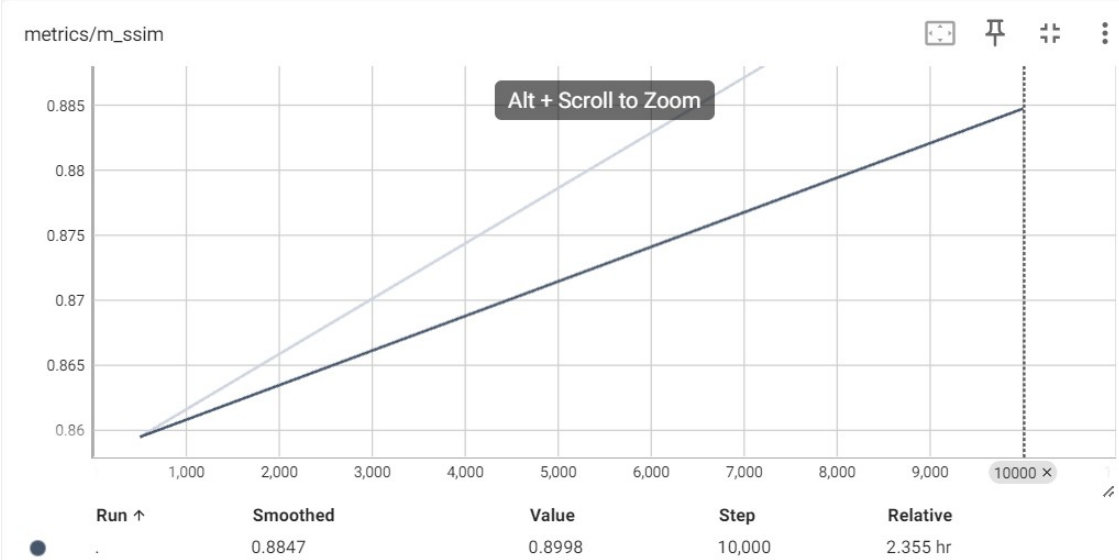
Sau khi huấn luyện dựa trên mô hình và dữ liệu trên, thu được biểu đồ thể hiện giá trị hàm mất mát và hai chỉ số SSIM và PSNR:



Hình 3.75. Biểu đồ minh họa loss của quá trình huấn luyện mô hình NAFNet



Hình 3.76. Biểu đồ minh họa chỉ số PSNR



Hình 3.77. Biểu đồ minh họa chỉ số SSIM

Đối với mô hình NAFNet này trong quá trình huấn luyện tổng cộng chỉ vỏn vẹn 15 epochs nhưng có thể thấy rằng hàm loss function đang có xu hướng giảm. Chứng tỏ rằng mô hình đang có xu hướng hoạt động tốt với tập dữ liệu GoPro trong quá trình khôi phục ảnh mờ. Bên cạnh đó, các chỉ số PSNR và SSIM của mô hình cũng có xu hướng tăng dần trong những epochs đầu và được ghi nhận ở một chỉ số tương đối tốt. Tuy nhiên, các chỉ số này vẫn chưa phải chỉ số lý tưởng cho một mô hình SOTA. Một số nguyên nhân dẫn đến việc các chỉ số PSNR và SSIM chưa đem lại kết quả lý tưởng có thể kể tới như:

- ***Tài nguyên bị hạn chế:*** Bởi vì tài nguyên huấn luyện trên Google Colaboratory bị giới hạn thời gian và không gian huấn luyện. Thay vì mô hình thực tế được tác giả huấn luyện song song với 8 gpu thì trong lần thử nghiệm này chỉ huấn luyện được với 1 gpu trong thời gian 6 giờ.
- ***Tập dữ liệu ảnh chưa đủ lớn:*** Tập dữ liệu GoPro thực tế có 2103 ảnh cho việc huấn luyện và với 1111 ảnh để kiểm thử. Nhưng trong thử nghiệm của phạm vi đồ án thì chỉ thử nghiệm với 701 ảnh vì giới hạn tài nguyên của Google Colaboratory.
- ***Số epochs huấn luyện không nhiều:*** Trong thực tế, mô hình được tác giả huấn luyện 50 epochs với 200000 iter thì thực nghiệm của đồ án chỉ huấn luyện 15 epochs với 20000 iter (giảm đi 10 lần so với huấn luyện thực tế).

Tuy việc huấn luyện không được diễn ra trong điều kiện thuận lợi nhưng nó cũng giúp cho sự quan sát đánh giá về độ hiệu quả và tốc độ của mô hình một cách khách quan rằng mô hình đang có xu hướng cải thiện với tập dữ liệu GoPro.

3.2.4. So sánh giữa các mô hình

Trong phạm vi thực hiện đồ án đã tiến hành thử nghiệm 3 mô hình, thuật toán để phục vụ cho tác vụ phát hiện ảnh mờ đó là mô hình MobileNet, thuật toán Fourier và các thuật toán học máy truyền thống Machine Learning. Bảng dưới đây đưa ra những so sánh, đánh giá ưu – nhược điểm của các phương pháp này nhằm kết luận phương án sử dụng để xây dựng hệ thống chương trình sau này.

	Ưu điểm	**Nhược điểm**	**Accuracy**
Thuật toán học máy Machine Learning	- Mô hình đơn giản, có dung lượng thấp, dễ dàng cài đặt. - Đạt hiệu quả chính xác cao trong thử nghiệm. - Tốc độ trả kết quả nhanh.	- Các mô hình học máy Machine Learning còn quá đơn giản, không tổng quát hết được dữ liệu huấn luyện nên có khả năng sẽ tồn tại overfitting	0.9175
Thuật toán	- Dễ dàng triển khai, cài	- Thuật toán triển khai	

Fourier	đặt thuật toán. - Dung lượng cài đặt thấp. - Tốc độ trả kết quả nhanh.	dựa trên việc đánh giá ngưỡng trung bình các pixel sau khi biến đổi nên không phản ánh chính xác tất cả các trường hợp. - Không có các chỉ số đánh giá đối với tập dữ liệu huấn luyện.	
Mô hình MobileNet	- Dễ dàng cài đặt thuật toán. - Tốc độ trả kết quả nhanh. - Đạt độ chính xác cao nhất lên tới 98%.	- Dung lượng mô hình lớn hơn hai phương pháp sử dụng thuật toán Fourier và mô hình học máy Machine Learning.	0.98

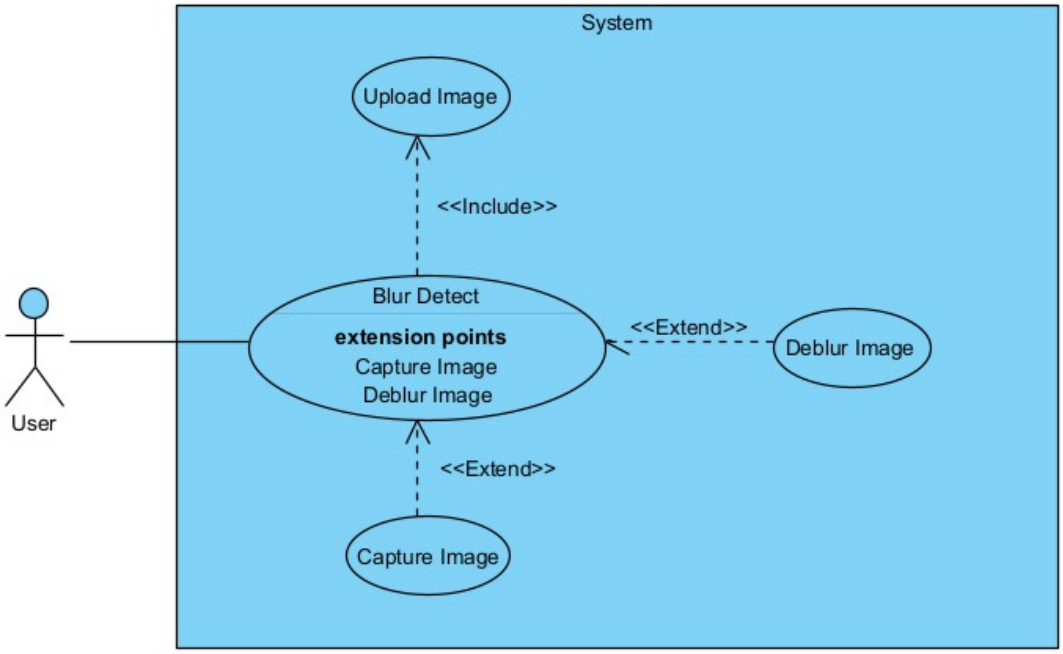
Bảng 3-3. So sánh các phương pháp trong tác vụ phát hiện ảnh mờ

Từ các kết quả đánh giá trên, có thể thấy mô hình học sâu MobileNet cụ thể là MobileNetV2 đạt hiệu quả chính xác cao nhất cho tác vụ phát hiện ảnh mờ. Mặc dù, mô hình vẫn tồn tại nhược điểm dung lượng mô hình lớn hơn so với hai phương pháp trên nhưng dung lượng mô hình là 9.9 MB nên vẫn ở mức cho phép khi cài đặt chương trình. Chính vì vậy, tác vụ phát hiện ảnh mờ sẽ áp dụng mô hình học sâu MobileNet để dự đoán các đầu vào ảnh. Ngoài ra, đồ án cũng sử dụng thêm thuật toán Fourier để phác thảo lại các cạnh biên, điểm ảnh nét trong quá trình phát hiện ảnh mờ.

3.3. Thiết kế hệ thống

3.3.1. Usecase tổng quan hệ thống

Hệ thống là một ứng dụng Android cho phép phát hiện và khôi phục một tấm ảnh bị mờ.

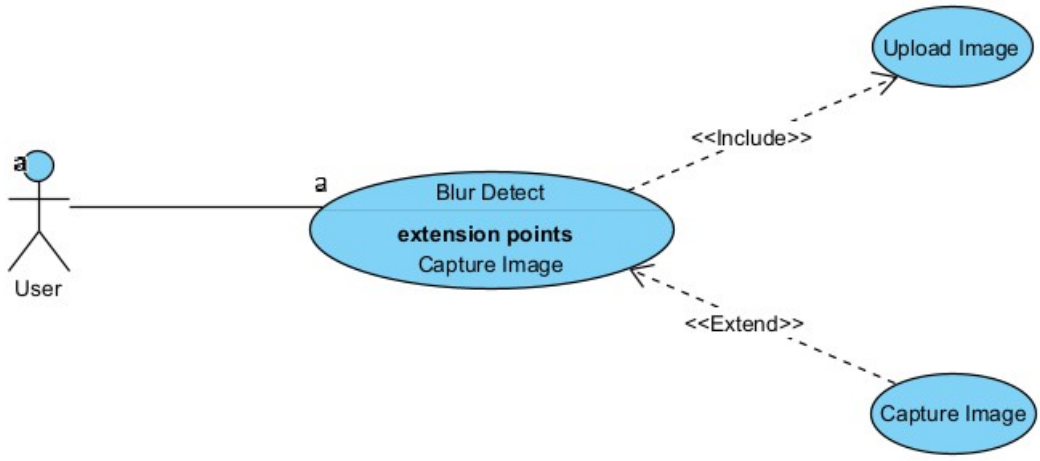


Hình 3.78. Usecase tổng quan hệ thống

Người sử dụng hệ thống có thể thực hiện hai chức năng chính bao gồm phát hiện và khôi phục hình ảnh bị mờ. Người dùng có thể chụp tấm ảnh và tải tấm ảnh được lựa chọn lên để hệ thống xử lý và trả về kết quả. Tấm ảnh sau khi được tải lên trước hết sẽ trả về kết quả dự đoán có phải là một tấm ảnh mờ hay không và phác thảo lên các cạnh biên của ảnh. Sau đó, người dùng có thể thực hiện tiếp chức năng thứ hai là khôi phục tấm ảnh đó. Tấm ảnh sẽ tiếp tục được hệ thống xử lý để khử mờ và trả về kết quả phía ứng dụng Android.

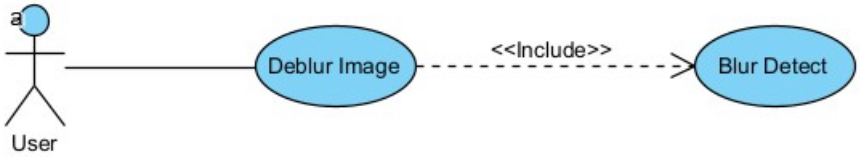
3.3.2. Phân rã biểu đồ usecase

❖ *Biểu đồ usecase phát hiện ảnh mờ*



Hình 3.79. Usecase phát hiện ảnh mờ

❖ *Biểu đồ usecase khôi phục ảnh mờ*



Hình 3.80. Usecase khôi phục ảnh mờ

3.3.3. Scenario

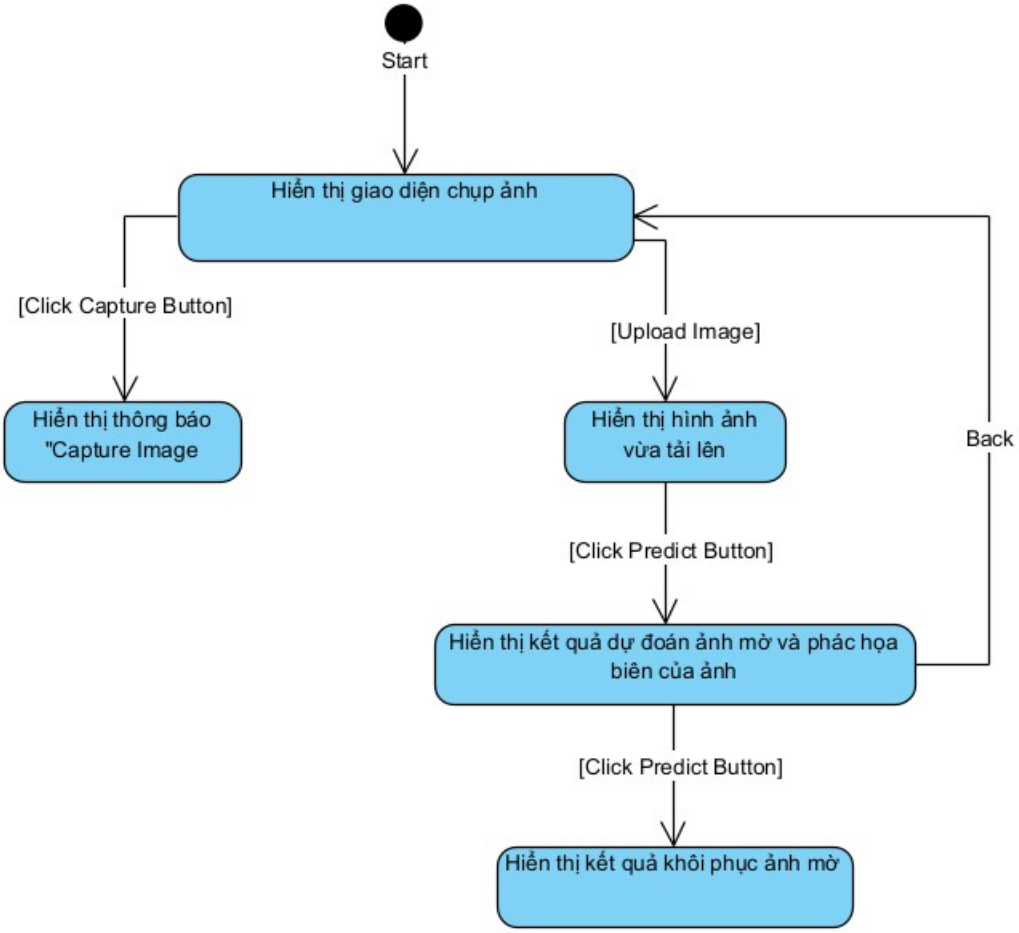
❖ *Chức năng phát hiện ảnh mờ*

Scenario	Phát hiện ảnh mờ
Actor	Người dùng
Post-condition	Người dùng nhận được kết quả dự đoán ảnh mờ
Main-events	1. Người dùng chụp ảnh từ thiết bị di động 2. Người dùng chọn vào biểu tượng thư viện ảnh trên ứng dụng 3. Người dùng lựa chọn hình ảnh cần dự đoán 4. Ứng dụng hiển thị lên tấm ảnh đã lựa chọn trên ImageView 5. Người dùng chọn nút "Predict" 6. Hệ thống xử lý trả về kết quả dự đoán hình ảnh mờ hay nét và kết quả phác thảo các cạnh biên của tấm ảnh trên ImageView

❖ *Chức năng khôi phục ảnh mờ*

Scenario	Khôi phục ảnh mờ
Actor	Người dùng
Pre-condition	Hệ thống đã phát hiện tấm ảnh mờ hay nét
Post-condition	Người dùng nhận được ảnh mới đã được khôi phục
Main-events	1. Người dùng tiếp tục sử dụng tấm ảnh đã dự đoán 2. Người dùng bấm nút chọn "Predict" 3. Hệ thống xử lý và trả về hình ảnh sau khi đã khôi phục để hiển thị lên ImageView

3.3.4. State Diagram



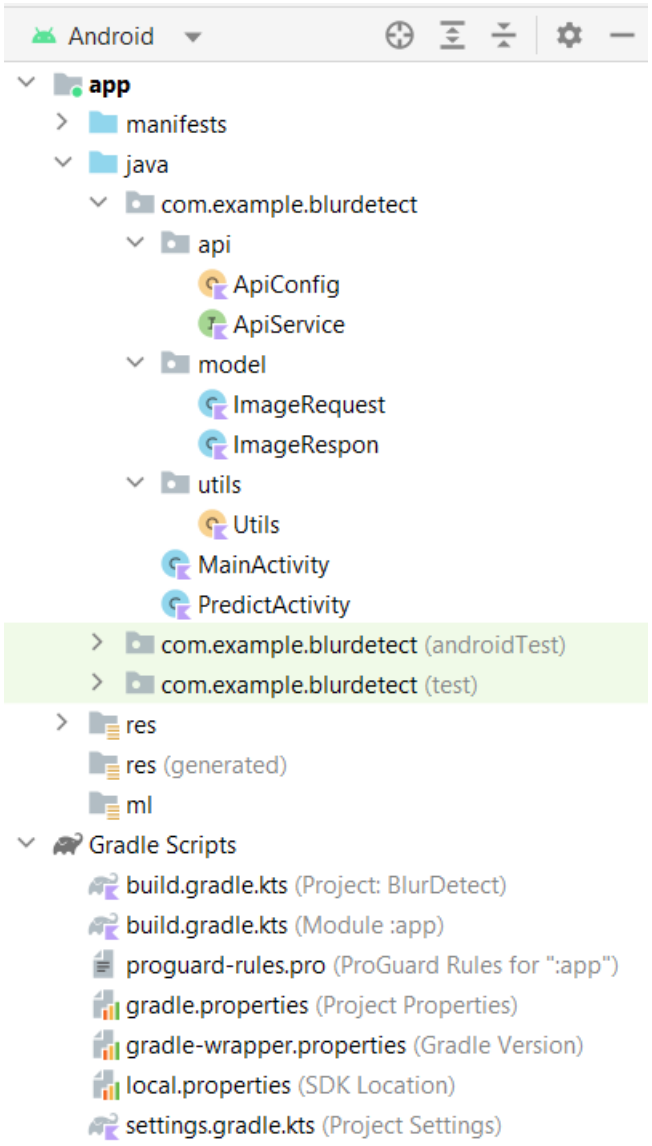
Hình 3.81. State Diagram

3.4. Xây dựng hệ thống

Sau giai đoạn huấn luyện thử nghiệm các mô hình học máy, thuật toán và mô hình học sâu để đưa ra các kết quả minh họa trên thì tiếp theo là giai đoạn tiến hành xây dựng hệ thống phần mềm cho ứng dụng Android phát hiện và khôi phục ảnh mờ.

3.4.1. Xây dựng ứng dụng

❖ *Cấu trúc thư mục*



Hình 3.82. Minh họa cấu trúc thư mục phía Client

api folder: bao gồm 1 Object để cấu hình cài đặt API và 1 Interface để viết các API Request lên server, trong đó:

- object ApiConfig: có nhiệm vụ cấu hình địa chỉ cơ bản để client giao tiếp với server và thời gian xử lý kết nối

```
package com.example.blurdetect.api

import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import java.util.concurrent.TimeUnit

object ApiConfig {
    private const val BASE_URL = "http://192.168.0.105:8000";
    val okHttpClient = OkHttpClient.Builder()
        .connectTimeout( timeout: 30, TimeUnit.SECONDS) // Thời gian timeout kết nối
        .readTimeout( timeout: 30, TimeUnit.SECONDS)    // Thời gian timeout đọc dữ liệu
        .writeTimeout( timeout: 30, TimeUnit.SECONDS)   // Thời gian timeout ghi dữ liệu
        .build()
    private val builder = Retrofit.Builder() Retrofit.Builder
        .baseUrl(BASE_URL) Retrofit.Builder
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create())

    val retrofit = builder.build()
//    val apiService: ApiService = retrofit.create(ApiService::class.java)
}
```

Hình 3.83. Object ApiConfig phía Client

- interface ApiService: viết các hàm request lên server bao gồm hai chức năng chính là phân loại ảnh và khử mờ ảnh. Đầu vào của các hàm này là ảnh của thiết bị để gửi lên server xử lý:

```
package com.example.blurdetect.api

import com.example.blurdetect.model.ImageRequest
import com.example.blurdetect.model.ImageRespon
import okhttp3.MultipartBody
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.Multipart
import retrofit2.http.POST
import retrofit2.http.Part

interface ApiService {
    @Multipart
    @POST("predict/")
    fun uploadImage(@Part image: MultipartBody.Part):Call<ImageRespon>

    @Multipart
    @POST("deblur/")
    fun deblurImage(@Part image: MultipartBody.Part):Call<ImageRespon>
}
```

Hình 3.84. interface ApiService phía Client

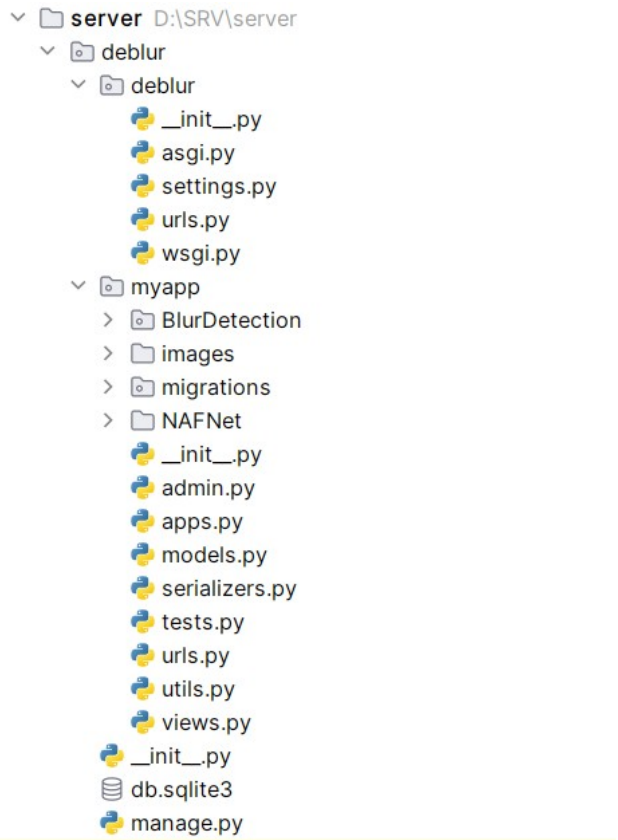
model folder: bao gồm 2 class đại diện cho model nhận và gửi dữ liệu ảnh

utils folder: có 1 object Utils có chức năng nhận đường dẫn ảnh để hiện thị lên ImageView

main: Hai class chính cho hai chức năng phân loại ảnh và khôi phục ảnh bao gồm các thao tác chụp ảnh, tải ảnh bên phía client.

3.4.2. Xây dựng server

❖ *Cấu trúc thư mục*



Hình 3.85. Cấu trúc thư mục phía Server

BlurDetection Folder: đây là folder áp dụng thuật toán Fourier được nêu trên để nhận diện một tấm ảnh và phác họa lại vùng mờ.

Images folder: sau khi client gửi ảnh lên, phía server sẽ nhận ảnh và lưu trữ trong folder này để xử lý hình ảnh. Các hình ảnh sau khi được xử lý sẽ được đẩy lên Cloudinary

NAFNet folder: đây là folder thực hiện triển khai model NAFNet để khôi phục ảnh mờ.

url.py: gọi đến urls.py chứa các đường dẫn trên server để client có thể giao tiếp cho server xử lý dữ liệu

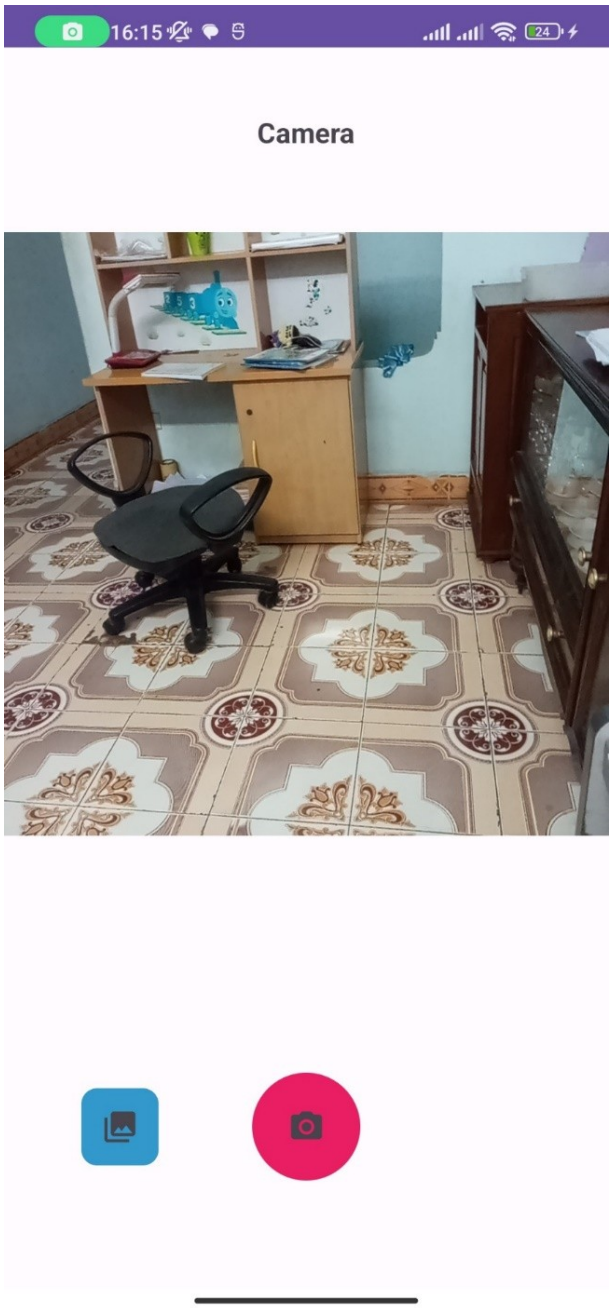
utils.py: đây là tập xử lý hình ảnh nhận vào của server, có nhiệm vụ chuyển đổi các hình ảnh nhận được thành các tensor phục vụ cho hai nhiệm vụ phân loại và khôi phục ảnh mờ

views.py: đây là tập hoạt động chính của server có nhiệm vụ xử lý dữ liệu hình ảnh để đẩy lên cloudinary rồi trả về phía client.

3.4.3. Kết quả chương trình

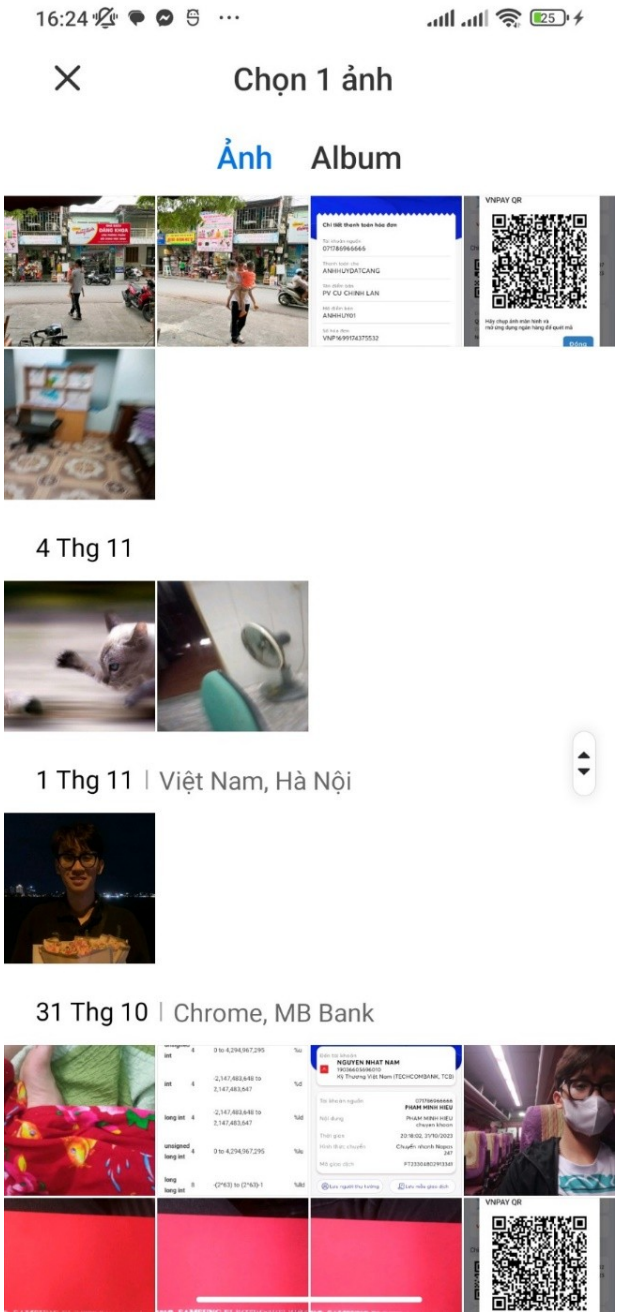
❖ *Chức năng phát hiện ảnh mờ*

- Bước 1: Người dùng có thể chụp ảnh hoặc tải ảnh lên để chương trình phân loại tấm ảnh đó là nét hay mờ.



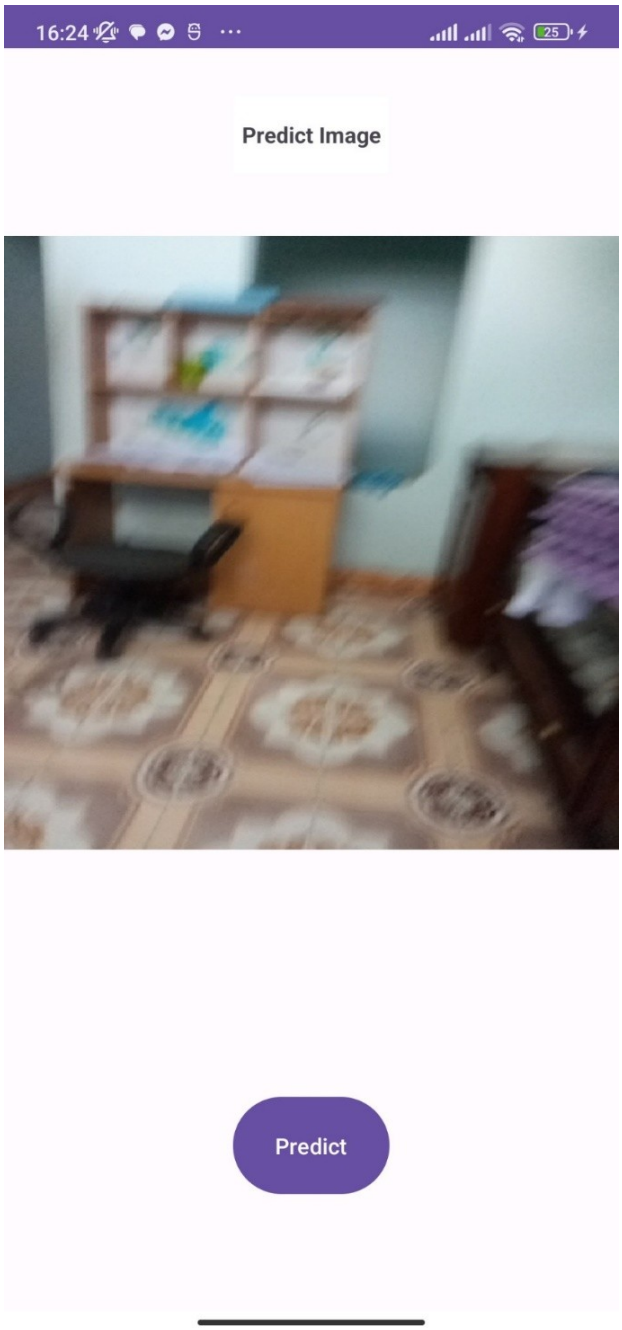
Hình 3.86. Minh họa màn hình ứng dụng chụp ảnh

- Bước 2: Người dung vào bộ sưu tập trên điện thoại để lựa chọn tấm ảnh cần dự đoán.



Hình 3.87. Minh họa màn hình ứng dụng cho việc tải ảnh lên

- Bước 3: Người dùng bấm nút "Predict" để chương trình tiến hành dự đoán hình ảnh



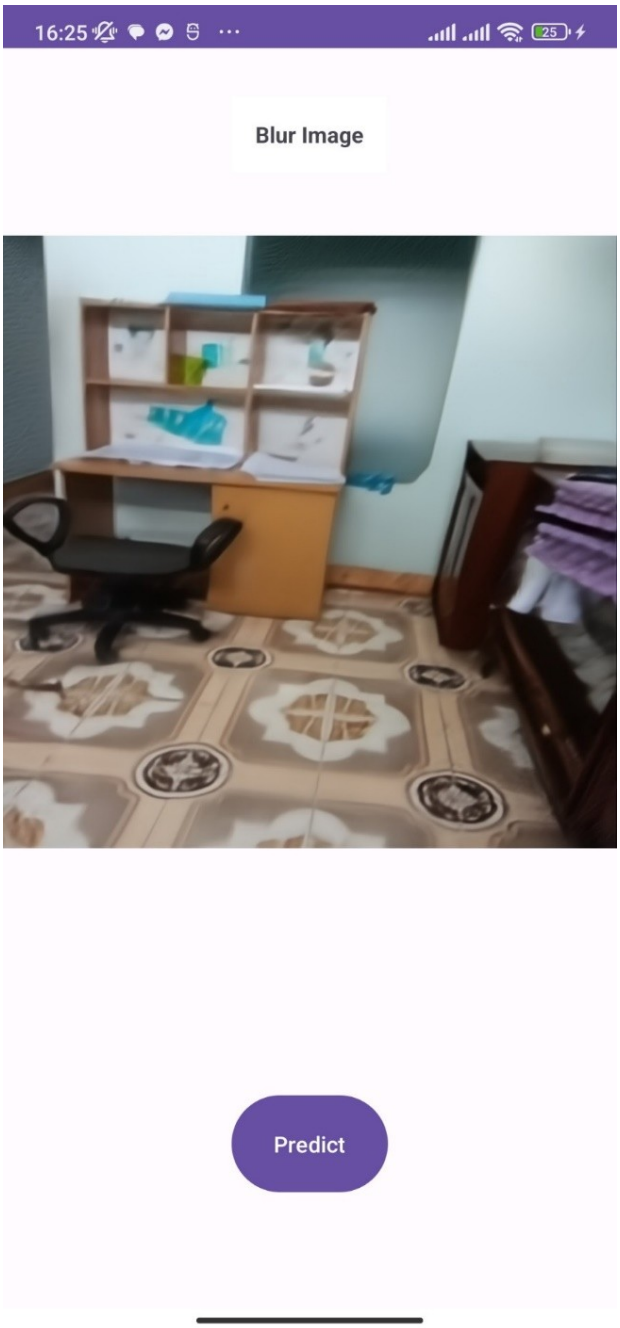
Hình 3.88. Minh họa màn hình sau khi chọn ảnh tải lên

- Bước 4: Chương trình hiển thị kết quả dự đoán và hình ảnh phân vùng mờ sau xử lý

Hình 3.89. Minh họa màn hình ứng dụng dự đoán và phân vùng ảnh mờ

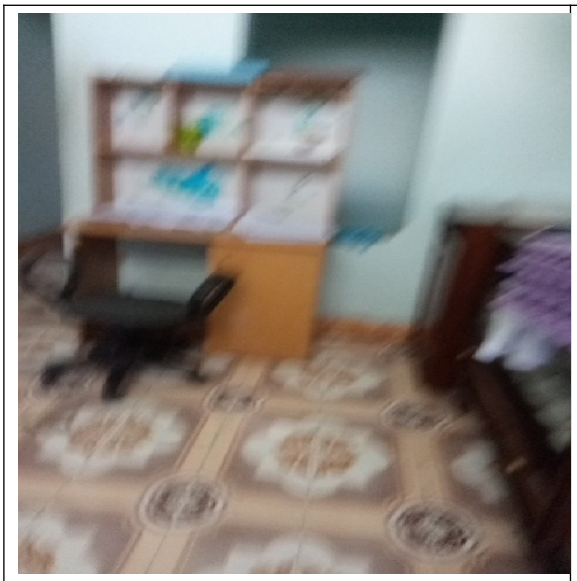
❖ *Chức năng khôi phục ảnh mờ*

- Bước 1: Người dùng tiếp tục sử dụng hình ảnh vừa được trả về ở bước phân loại hình ảnh và bấm vào nút "Predict" tiếp để khử mờ ảnh.
- Bước 2: Chương trình xử lý sau vài giây và trả về kết quả hình ảnh được khôi phục do bị mờ.



Hình 3.90. Minh họa màn hình ứng dụng sau khi khôi phục ảnh mờ

Như vậy, sau khi khử mờ thành công chương trình sẽ trả cho chúng ta một tấm ảnh mới nét hơn ảnh cũ ban đầu. Dưới đây là so sánh giữa hai hình ảnh ban đầu và sau khi được khử mờ thành công



Ảnh bị mờ | Ảnh sau khi khôi phục

Hình 3.91. So sánh hình ảnh trước và sau khi khôi phục

❖ ***Kết luận chương***

Chương 3 "Xây dựng chương trình" cũng là chương cuối cùng của đồ án. Nội dung chương đã làm rõ các bước xây dựng một mô hình và tiến hành thử nghiệm cũng như đưa ra các đánh giá, kết luận trực quan. Cuối cùng, chương 3 kết thúc với nội dung xây dựng chương trình và đưa ra các hình ảnh minh họa sự hoạt động của hệ thống phần mềm này.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

❖ ***Kết luận đồ án***

Xuyên suốt quá trình học hỏi, nghiên cứu và xây dựng đề tài đồ án **"Áp dụng mô hình học sâu triển khai ứng dụng Android phát hiện và khôi phục ảnh mờ"** của mình em đã thu nhận được rất nhiều kiến thức và trải nghiệm mới giúp bản thân mình cải thiện, nâng cao thêm sự hiểu biết về một lĩnh vực mới liên quan đến xử lý ảnh. Từ những trải nghiệm thực tế này tạo cho em nhiều cơ hội học hỏi mới như việc tìm hiểu về xử lý ảnh, các khái niệm và các thao tác xử lý ảnh. Bên cạnh tìm hiểu các kiến thức cơ bản về xử lý ảnh em cũng có cơ hội nghiên cứu thêm về mạng nơ – ron tích chập CNN, các khối phi tuyến tính như ReLU, GeLU [16], các hàm kích học như Softmax hay các chỉ số đánh giá hiệu năng của một mô hình cho bài toán phân loại và chỉ số đánh giá khác như SSIM và PSNR. Ngoài việc tìm hiểu ra, em cũng có cơ hội thử sức lập trình với các thuật toán học máy cơ bản trên thư viện scikit-learn như thuật toán K-Nearest Neighbors, thuật toán Decision Tree, thuật toán Naive Bayes và thuật toán Random Forest. Không chỉ dừng lại ở các thuật toán cơ bản, em cũng có cơ hội tìm hiểu thêm nhiều mô hình Deep Learning khác cho bài toán phát hiện và khôi phục ảnh mờ như mô hình MobileNet, mô hình NAFNet cùng các bài báo khoa học liên quan đến hai mô hình này. Bản thân em cũng có thêm nhiều khả năng tìm kiếm, sưu tầm các kỹ thuật xử lý ảnh và trau dồi thêm khả năng lập trình cũng như các kiến thức về xây dựng phần mềm. Từ đó em tiến hành thử nghiệm các mô hình trên các thuật toán học máy, mô hình học sâu để đưa ra đánh giá. Cuối cùng em tập trung phát triển một ứng dụng cơ bản để phát hiện và khôi phục ảnh bị mờ do chuyển động hoặc rung lắc. Từ ứng dụng này, trong thực tế có thể ứng dụng cho rất nhiều lĩnh vực cần đến xử lý ảnh mờ như y tế, an toàn giao thông, ngân hàng,…Bên cạnh các kết quả đạt được cũng tồn tại những điểm cần cải thiện như tốc độ khử mờ hình ảnh trên ứng dụng. Đôi khi ứng dụng nhận diện sai kết quả mặc dù tỉ lệ chính xác của mô hình khá cao. Ngoài ra, ứng dụng chỉ tập trung vào ảnh mờ toàn phần bị rung lắc nên vẫn xảy ra sai xót đối với các loại ảnh mờ khác như ảnh chụp tiêu điểm.

❖ ***Hướng phát triển trong tương lai***

Qua những thành quả và hạn chế của quá trình thực hiện xây dựng hệ thống, cho thấy việc nghiên cứu những công nghệ áp dụng cho đề tài đòi hỏi một quá trình thực hiện lâu dài và nhiều công sức. Trong phạm vi đồ án với một thời gian thực hiện khoảng gần 3 tháng, em nhận ra hệ thống còn cần phải cải thiện nhiều điểm để phát triển như:

- Nâng cao tỉ lệ nhận diện chính xác với các loại ảnh mờ khác.
- Thực hiện phân vùng mờ đối với các tấm ảnh bị mờ một phần.
- Cải thiện tốc độ khôi phục ảnh mờ của ứng dụng.
- Nâng cao chất lượng bộ dữ liệu và mô hình để tối đa hóa hiệu năng hướng tới một thuật toán hoạt động theo thời gian thực.

TÀI LIỆU THAM KHẢO

[1] L. Zhaorong, L. Renting and J. Jiaya, "Image Partial Blur Detection and Classification," p. 1, 2008.

[2] T. T. Nguyen, "Deep Learning cơ bản," 24 03 2019. [Online]. Available: https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/. [Accessed 15 12 2023].

[3] R. C. Gonzalez and R. E. Woods, in *Digital Image Processing*, 2018.

[4] H. Nguyen, "Vietnix," 21 04 2022. [Online]. Available: https://vietnix.vn/deep-learning-la-gi/. [Accessed 15 12 2023].

[5] L. Xiaoyang and Z. Zhigang, "Memristor crossbar architectures for implementing deep neural," p. 9, 2021.

[6] V. H. Tiep, "Machine Learning cơ bản," 21 01 2017. [Online]. Available: https://machinelearningcoban.com/2017/01/21/perceptron/. [Accessed 15 12 2023].

[7] A. Saxena, "Medium," 01 06 2020. [Online]. Available: https://towardsdatascience.com/building-a-simple-neural-network-from-scratch-a5c6b2eb0c34. [Accessed 12 12 2023].

[8] C. E. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of Trends in," p. 7, 24 02 2018.

[9] S. Sharma, S. Simone and A. Anidhya, "ACTIVATION FUNCTIONS IN NEURAL," p. 313, 2020.

[10] A. Amidi and S. Amidi, "Stanford University," [Online]. Available: https://stanford.edu/~shervine/l/vi/teaching/cs-230/cheatsheet-convolutional-neural-networks. [Accessed 12 12 2023].

[11] V. H. Tiep, "Machine Learning cơ bản," 03 01 2018. [Online]. Available: https://machinelearningcoban.com/2017/08/31/evaluation/. [Accessed 15 12 2023].

[12] S. Umme, A. Morium and U. S. Mohammad, "Image Quality Assessment through FSIM, SSIM, MSE and PSNR - A Comparative Study," 2019.

[13] M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreeetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision," 2017.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019.

[15] L. Chen, X. Chu, X. Zhang and J. Sun, "Simple Baselines for Image Restoration," 2022.

[16] D. Hendrycks and K. Gimpel, "GAUSSIAN ERROR LINEAR UNITS,"

2021.